Telink BT/BLE Controller SDK Developer Handbook

Functional Specifications

AN-21122300-E1

Ver.1.0.0

2021/12/23

Keyword

Bluetooth, Bluetooth LE, Controller, SDK

Brief

This document presents the guide on the functional specifications of Telink BT/BLE dual-mode controller SDK.

Published by

Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,

Zhangjiang Hi-Tech Park, Shanghai, China

Legal Disclaimer

This document is provided as-is. Telink Semiconductor reserves the right to make improvements without further notice to this document or any products herein. This document may contain technical inaccuracies or typographical errors. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein.

Information

For further information on the technology, product and business term, please contact Telink Semiconductor Company (www.telink-semi.com).

For sales or technical support, please send email to the address of:

telinksales@telink-semi.com

telinksupport@telink-semi.com

Revision History

Version	Change Description	Date	Author
V1.0.0	Initial release.	2021/12	Peng.QI, C.LIU

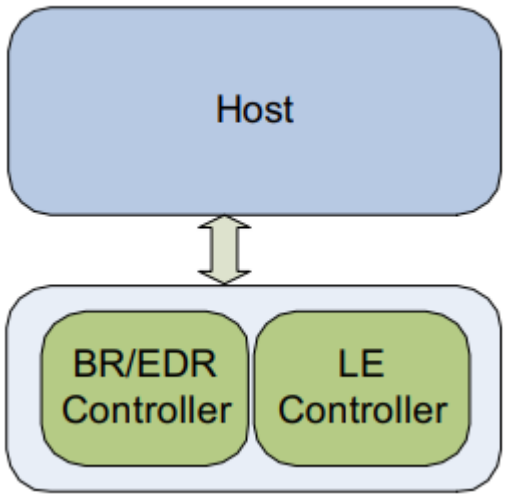
Contents

1. Overview

1.1 Document Overview

This document is mainly a functional description of Telink BT/BLE dual-mode controller SDK, which adapts to different host through the HCI interface, supports the standard HCI protocol and HCI flow control, hardware supports UART and USB module as the transmission interface.

Figure 1-1 Transfer between host and controller



	LE only	BT only	BT/BLE
Mode	√	√	√

1.2 SDK Structure

Figure 1-2 SDK structure overview

main.c

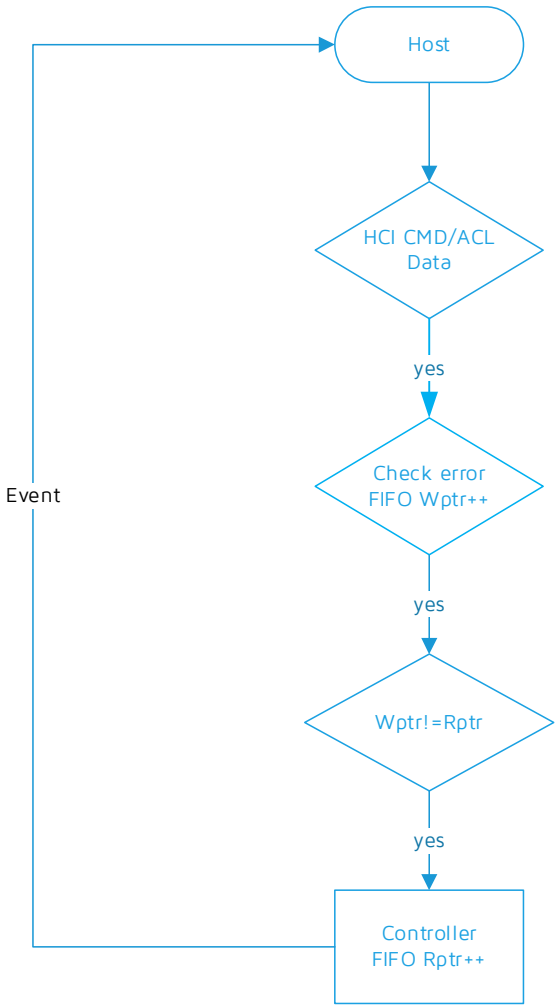
```
{
    sys_init(DCDC_1P4_LDO_1P8,VBAT_MAX_VALUE_GREATER_THAN_3V6);

    clock_init(PLL_CLK_192M, PAD_PLL_DIV, PLL_DIV4_TO_CCLK, CCLK_DIV2_TO_HCLK, HCLK_DIV1_TO_PCLK, CCLK_TO_MSPI_CLK);
    /* random number generator must be initiated here( in the beginning of user_init_nromal).
     * When deepSleep retention wakeUp, no need initialize again */
    random_generator_init();  //this is must

    intcntl_init();

    io_debug_init();

#if (ENABLE_PRINT||ENABLE_VCD)
    debug_init();
#endif

#if(ENABLE_BT_CLASSIC||ENABLE_LE)
    hci_transport_init(HCI_TRANSPORT_MODE);
#endif

#if(ENABLE_BT_CLASSIC||ENABLE_LE)
    btble_schedule_init();
#endif

#if ENABLE_LE
    ble_init();
#endif

#if ENABLE_BT_CLASSIC
    bt_init();
#endif

    user_init();

    core_enable_interrupt();

    while (1) {
        main_loop();
```

- sys_init: system initialization, configure system power mode and voltage level.
- clock_init: clock initialization.
- random_generator_init: random number generation module initialization.
- intcntl_init: interrupt table initialization.
- io_debug_init: GPIO debugging initialization, general GPIO function, user can configure by themselves.
- debug_init: system debug initialization, developers can use to print log.
- hci_transport_init: HCI module initialization, used to configure the transfer method.
- btble_schedule_init: system scheduling initialization, the system uses it to schedule the operation of various tasks.

- ble_init: ble module initialization.
- bt_init: bt module initialization.
- user_init: module reserved for developer use.
- core_enable_interrupt: after all the previous initialization, the global interrupt is enabled and the system is ready to enter the running state.
- main_loop: system loop, where various tasks are handled.

1.3 HCI Flow

Figure 1-3 HCI flow



2. Controller Configuration

2.1 LE Link Configuration

This section is about the configuration of LE part, whether to enable LE or not, and the number of connections when LE is master and slave.

```
/* LE */
#define ENABLE_LE                   1
#define MASTER_MAX_NUM              2
#define SLAVE_MAX_NUM               1
#define BLE_MAX_CONN_NUM            (MASTER_MAX_NUM + SLAVE_MAX_NUM)
```

To disable the LE function completely, set ENABLE_LE to 0.

2.2 BT Link Configuration

This section is about the configuration of the BT part, note that if only one channel of BT connection is needed, BT_ACL_LINK_NUM can be set to 1.

```
/* CLASSIC BLUETOOTH*/
#define ENABLE_BT_CLASSIC           1

#define BT_ACL_LINK_NUM             1
```

The BT module contains functional modules in registration mode, and customers can choose whether to register this function according to their needs.

Figure 2-1 BT link configuration

*app_bt.c stack_config.h

```
102
103 void btc_module_init(void)
104 {
105     // AFH
106     btc_afh_register_module(&env_afh[0]);
107     // AFH CLS
108     btc_afh_cls_register_module();
109     // ROLE SWITCH
110     btc_rsw_register_module(&env_rsw[0]);
111
112     // LEGACY PAIR
113     btc_legacy_pair_register_module();
114     // SECURE SIMPLE PAIR
115     btc_ssp_register_module(&env_ssp[0]);
116     // LEGACY AUTHEN
117     btc_legacy_authen_register_module();
118     // SECURE AUTHEN
119
120     // ENCRYPTION
121     btc_enc_register_module(&env_enc[0]);
122     // ESCO
123 //    btc_sco_register_module(&env_sco[0]);
124     // SNIFF
125 //    btc_sniff_register_module(&env_sniff[0]);
126     // LOW POWER
127
128     //TEST MODE
129 //    btc_test_mode_register_module(&env_test_mode[0]);
130 }
131
132
```

2.3 HCI Transport Configuration

This section is about the configuration of the HCI part, where the user chooses which method to use as the transport.

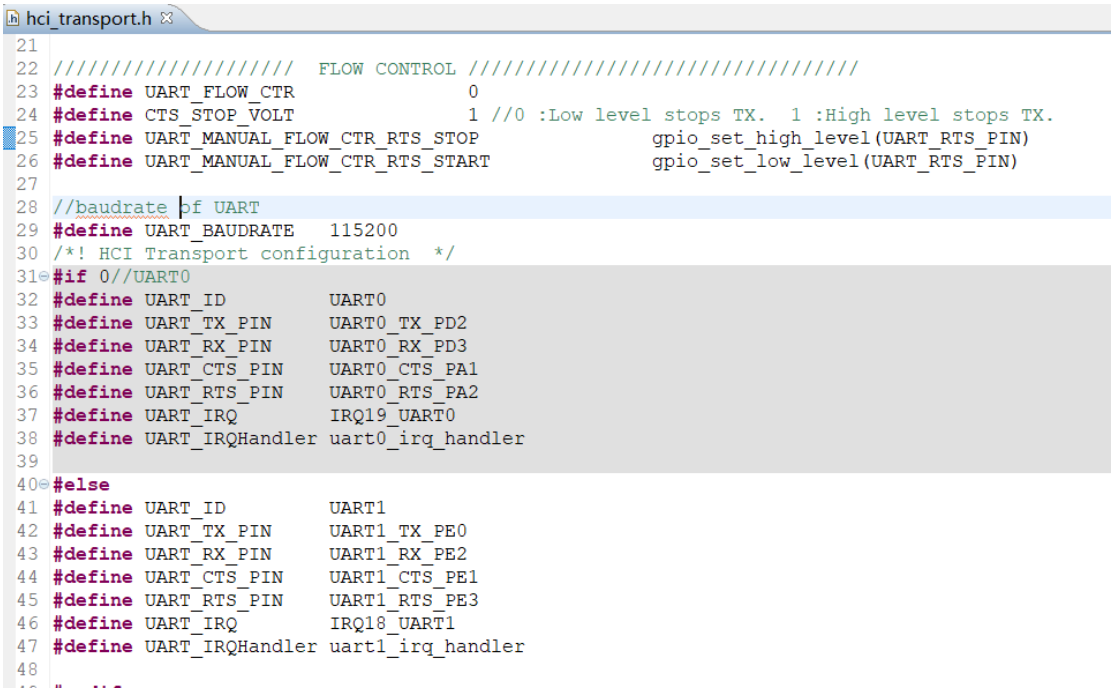
```
/* HCI TRANSPORT*/
#define ENABLE_HCI_TRANSPORT        1
#define HCI_TRANSPORT_MODE          2//1-usb bulk,2-uart
```

2.3.1 UART Transport Configuration

The configuration of UART as HCI transport requires that the docking device on the host end should support UART flow control.

The UART transport configuration Includes baud rate configuration, port configuration, and pin configuration.

Figure 2-2 UART transport configuration



hci_transport.h

```
21
22 ////////////////////  FLOW CONTROL //////////////////////////////////
23 #define UART_FLOW_CTR                0
24 #define CTS_STOP_VOLT                1 //0 :Low level stops TX.  1 :High level stops TX.
25 #define UART_MANUAL_FLOW_CTR_RTS_STOP               gpio_set_high_level(UART_RTS_PIN)
26 #define UART_MANUAL_FLOW_CTR_RTS_START              gpio_set_low_level(UART_RTS_PIN)
27
28 //baudrate of UART
29 #define UART_BAUDRATE   115200
30 /*! HCI Transport configuration  */
31 #if 0//UART0
32 #define UART_ID          UART0
33 #define UART_TX_PIN      UART0_TX_PD2
34 #define UART_RX_PIN      UART0_RX_PD3
35 #define UART_CTS_PIN     UART0_CTS_PA1
36 #define UART_RTS_PIN     UART0_RTS_PA2
37 #define UART_IRQ         IRQ19_UART0
38 #define UART_IRQHandler uart0_irq_handler
39
40 #else
41 #define UART_ID          UART1
42 #define UART_TX_PIN      UART1_TX_PE0
43 #define UART_RX_PIN      UART1_RX_PE2
44 #define UART_CTS_PIN     UART1_CTS_PE1
45 #define UART_RTS_PIN     UART1_RTS_PE3
46 #define UART_IRQ         IRQ18_UART1
47 #define UART_IRQHandler uart1_irq_handler
48
```

2.3.2 USB Transport Configuration

The configuration of USB is as follows. Note that the only way to print in this mode is to use the UART debug method.

```
#define ENABLE_HCI_TRANSPORT        1
#define HCI_TRANSPORT_MODE          1//1-usb bulk,2-uart
```

3. Debug and Print

3.1 USB Print log

This mode needs to be used with Telink proprietary software. There is a conflict in USB HCI mode, and it can not use this mode. It can only be used in UART HCI mode, the demonstration is as follows.

```
#define PRINT_MODE                  0 //usb print mode
```

Figure 3-1 USB print log

```
17:54:27.807  <53> HCI-> Event:         04 0e 09 01 27 0c 00 11  00 00 00 00
17:54:28.241  <54> <<<_CMD_CMPT>        04 070e         01 05 14 00 11 00 00
17:54:28.242  <55> dma_offset&readmasize:       04 00 00 00 ff 03 00 00  02 00 00 00 00 00 00 00
17:54:28.242  <56> fifo_hci_data :
        06 00 00 00 01 05 14 02  11 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.242  <57> packet_len :         06 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.243  <58> fifo_hci_data_p :
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.243  <59> complete_hci_data from_host:
17:54:28.243  <60> <<<_CMD_CMPT>        04 090e         01 27 0c 00 11 00 00 00  00
17:54:28.243  <61> @ bt-->hci_exec data(8)_id(4)_fucntion:      01 05 14 02 11 00 00 00  f5 00 00 00 68
17:54:28.244  <62> @ HCI_RD_RSSI_CMD_OPCODE_1405_f5:    01 05 14 02 11 00
17:54:28.244  <63> HCI-> Event:         04 0e 07 01 05 14 00 11  00 00
17:54:28.244  <64> dma_offset&readmasize:       04 00 00 00 ff 03 00 00  02 00 00 00 00 00 00 00
17:54:28.244  <65> fifo_hci_data :
        06 00 00 00 01 27 0c 02  11 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.244  <66> packet_len :         06 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.245  <67> fifo_hci_data_p :
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
17:54:28.245  <68> complete_hci_data from_host:
17:54:28.245  <69> @ bt-->hci_exec data(8)_id(4)_fucntion:      01 27 0c 02 11 00 00 00  83 00 00 00 64
17:54:28.245  <70> @ HCI_RD_AUTO_FLUSH_TO_CMD_OPCODE_0c27_83:   01 27 0c 02 11 00
17:54:28.245  <71> HCI-> Event:         04 0e 09 01 27 0c 00 11  00 00 00 00
```

3.2 UART Print log

The print log mode can be used under UART/USB HCI.

```
#define PRINT_MODE                  1//uart print mode
```

This debug method uses the UART DMA method and allows the user to configure the module pins and baud rate.

Figure 3-2 UART print configuration

```
main.c   app_bt.c   app.c   hci_transport.c   hci_transport.h   stack_config.h

#ifndef         PRINT_MODE
#define         PRINT_MODE          0// 0->USB,1->UART
#endif

#endif

#if ((ENABLE_PRINT==1)&&(PRINT_MODE == 1))

#define UART_PRINTF_BAUDRATE        2000000//2M

#define UART_PRINTF_ID              UART0

#define UART_PRINTF_IO_TX           UART0_TX_PA3

#define UART_PRINTF_IO_RX           UART0_RX_PA4

#define ENABLE_VCD                  0
```

It is important to note that when using UART HCI, never configure the pins of UART debug to be the same with that of UART HCI.

Figure 3-3 UART print log

```
@ bt—>hci_exec data(8)_id(4)_fucntion:01 03 0c 00 00 00 00 00 5f 00 00 00 e0 7e 02 20
HCI_RESET_CMD_OPCODE_0c03_5f:01 03 0c 00
pscan_win:40 00
pscan_intv:ac 00
HCI-> Eent:04 0e 04 01 03 0c 00
dma_offset&readmasize:04 00 00 00 ff 03 00 00 00 00 00 00 00 00 00 00
fifo_hci_data :04 00 00 00 01 05 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
packet_len :04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
fifo_hci_data_p :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
complete_hci_data from_host:
@ bt—>hci_exec data(8)_id(4)_fucntion:01 05 10 00 00 00 00 00 e5 00 00 00 ec 69 02 20
@ HCI_RD_BUFF_SIZE_CMD_OPCODE_1005_e5:01 05 10 00
HCI-> Event:04 0e 0b 01 05 10 00 bc 02 40 04 00 08 00
dma_offset&readmasize:08 00 00 00 fe 03 00 00 03 00 00 00 00 00 00 00
fifo_hci_data :0b 00 00 00 01 33 0c 07 9b 06 ff 14 00 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
packet_len :0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00
fifo_hci_data_p :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
complete_hci_data from_host:
@ bt—>hci_exec data(8)_id(4)_fucntion:01 33 0c 07 9b 06 ff 14 8f 00 00 00 30 76 02 20
@ HCI_HOST_BUF_SIZE_CMD_OPCODE_0c33_8f:01 33 0c 07 9b 06 ff 14 00 0a 00
HCI-> Event:04 0e 04 01 33 0c 00
dma_offset&readmasize:04 00 00 00 ff 03 00 00 00 00 00 00 00 00 00 00
fifo_hci_data :04 00 00 00 01 01 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
packet_len :04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
fifo_hci_data_p :00 00 00 00 00 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
complete_hci_data from_host:
@ bt—>hci_exec data(8)_id(4)_fucntion:01 01 10 00 00 00 00 00 e1 00 00 00 78 6b 02 20
@ HCI_RD_LOCAL_VER_INFO_CMD_OPCODE_1001_e1:01 01 10 00
HCI-> Event:04 0e 0c 01 01 10 00 0b 0c 00 0b 11 02 0b 00
dma_offset&readmasize:04 00 00 00 ff 03 00 00 00 00 00 00 00 00 00 00
fifo_hci_data :04 00 00 00 01 09 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
packet_len :04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
fifo_hci_data_p :00 00 00 00 00 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
complete_hci_data from_host:
@ bt—>hci_exec data(8)_id(4)_fucntion:01 09 10 00 00 00 00 00 e9 00 00 00 90 69 02 20
@ HCI_RD_BD_ADDR_CMD_OPCODE_1009_e9:01 09 10 00
HCI-> Event:04 0e 0a 01 09 10 00 2e c5 c9 60 b8 03
dma_offset&readmasize:04 00 00 00 ff 03 00 00 00 00 00 00 00 00 00 00
```

4. User-defined HCI CMD

The SDK reserves the callback interface to implement the user's own HCI custom functions. You need to register the corresponding custom functions in accordance with the OCF value in the following table.

Table 4-1 OCF table

OGF	OCF
0x3F	0x08
	0x09
	0x0a
	0x0b

Demo:

When the return value is 0, the registration is correct, otherwise the registration fails.

```
int hci_vendor_spec_register_cb_port(hci_cmd_vendor_spec_callback_t cb,u8 vs_opcode_num);
```

Figure 4-1 Demo of registration

app.c

```
47 }
48 void my_vs_test_func(u8 *para)
49 {
50     gpio_toggle(GPIO_D);
51 }
52
53
54 void user_init(void)
55 {
56     hci_vendor_spec_register_cb_port(my_vs_test_func,0x08);
57 }
```