Renesas USB MCU

R01AN0664EJ0215
Rev.2.15
Mar 28, 2016

USB Host Human Interface Device Class Driver (HHID) using Basic Mini Firmware

Introduction

This document is an application note describing use of the USB Host Human Interface Device Class Driver (HHID) built using the USB Basic Mini Firmware of the Renesas USB MCU.

Target Device

RL78/G1C, R8C/3MK, R8C/34K

This program can be used with other microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

This program has been evaluated using the corresponding MCU’s Renesas Starter Kit board.

Contents

1. Overview

This application note describes the USB Host Human Interface Device Class Driver (HHID) and the sample application using USB Basic Mini Firmware (refer to the Chapter 1.2).

1.1 Functions and Features

The USB Host Human Interface Device Class Driver (HHID) conforms to the USB Human Interface Device Class specification (HID from now on and description). It and enables communication with a HID peripheral device.

This class driver is intended to be used in combination with the USB Basic Mini Firmware provided from Renesas Electronics.

1.2 Related Documents

1. Universal Serial Bus Revision 2.0 specification
2. USB Class Definitions for Human Interface Devices Version 1.1
3. HID Usage Tables Version 1.1
   [http://www.usb.org/developers/docs/]
4. User's Manual: Hardware
5. USB Basic Mini Firmware Application Note (Document No.R01AN0326EJ)

Available from the Renesas Electronics Website

・ Renesas Electronics Website
   http://www.renesas.com/
・ USB Devices Page
   http://www.renesas.com/prod/usb/

1.3 Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

API	:	Application Program Interface
APL	:	Application program
cstd	:	Prefix for peripheral & host common function of USB-BASIC-F/W
Data Transfer	:	Generic name of Control transfer, Bulk transfer and Interrupt transfer
HCD	:	Host control driver of USB-BASIC-F/W
HDCD	:	Host device class driver (device driver and USB class driver)
HEW	:	High-performance Embedded Workshop
HHID	:	Host human interface device
HID	:	Human interface device class
HM	:	Hardware Manual
hstd	:	Prefix for host function of USB-BASIC-F/W
KBD	:	Keyboard device
MGR	:	Peripheral device state manager of HCD
MSE	:	Mouse device
PP	:	Pre-processed definition
RSK	:	Renesas Starter Kit
Scheduler	:	Used to schedule functions, like a simplified OS
Scheduler Macro	:	Used to call a scheduler function
SW1/SW2/SW3	:	User switches on RSK
Task	:	Processing unit
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB-BASIC-F/W (Peripheral & Host USB Basic Mini Firmware(USB low level) for Renesas USB MCU)

1.4 How to Read This Document

To run the demo, start by reading “USB Host Human Interface Device Class Driver (HHID) Installation Guide for USB Basic Mini Firmware”.

This document is not intended for reading straight through. Use it first to gain acquaintance with the package, then to look up information on functionality and interfaces as needed for your particular solution.

Chapter 5 explains how the default host HID demo application works. You will change this to create your own solution.

Understand how all code modules are divided into tasks, and that these tasks pass messages to one another. This is so that functions (tasks) can execute in the order determined by a scheduler and not strictly in a predetermined order. This way more important tasks can have priority. This plus the use of a function callback mechanism enables the USB code to be non-blocking. The task mechanism is described in Chapter 1.2 above "USB Basic Mini Firmware Application Note".

All HID tasks are listed in Chapter 4.4.

2. Register Class Driver

A class driver must be registered with the USB-BASIC-F/W. Please consult function *usb_hapl_registration()* in *r_usb_hhid_apl*.on how to register a class driver with USB-BASIC-F/W. For details, please refer to USB Basic Mini Firmware application note.

3. Operating Confirmation Environment

3.1 Compiler

The compilers which is used for the operating confirmation are follows.

a. CA78K0R Compiler V.1.71

b. CC-RL Compiler V.1.01

c. IAR C/C++ Compiler for RL78 version 2.10.4

d. KPIT GNURL78-ELF v15.02

e. C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Release 00

3.2 Evaluation Board

The evaluation boards which is used for the operating confirmation are follows.

a. Renesas Starter Kit for RL78/G1C (Product No: R0K5010JGC001BR)

b. R8C/34K Group USB Host Evaluation Board (Product No: R0K5R8C34DK2HBR)

4. Software Configuration

4.1 Module Configuration

The HHID comprises the HID class driver and the device drivers for mouse and keyboard.

Figure 4.1 shows the structure of the HHID software modules. Table 4-1 lists the modules and an overview of each.

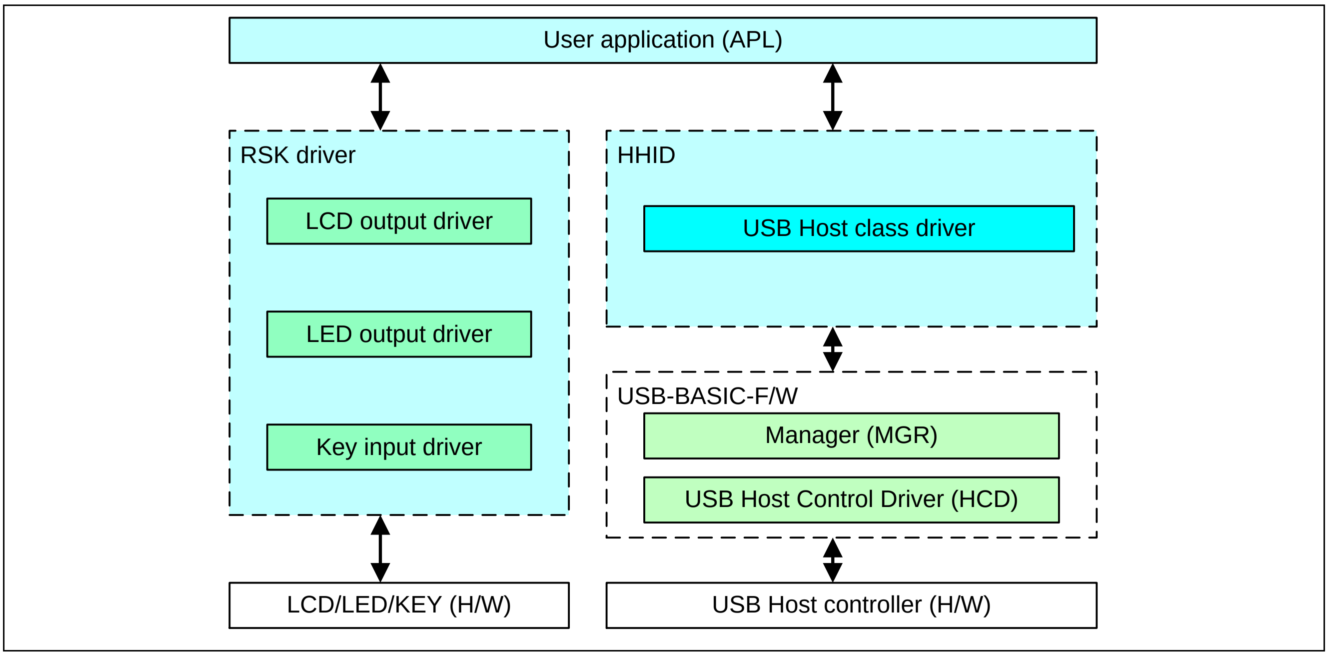

Figure 4.1 Module Structure

Table 4-1 Module Function Descriptions

Module Name	Description	Notes
APL	User application program. Board (RSK) switches initiate communication with attached HID devices and control suspend/resume. The LCD displays the information received from the HID device.	Created by the customer.
HHID	The registered device class driver checks operation of the connected device. The USB-BASIC-F/W checks whether the connected device enables for HHID. The following data transfers are requested of USB-BASIC-F/W by the APL. 1) Control of connected device by HID requests 2) Data transfer with connected device Transfer results are notified to APL by a callback function.	
USB-BASIC-F/W	USB Basic Mini Firmware (Host Hardware Control & Device state Management)	

4.2 Overview of Application Program Functions

The main functions of the host demo application:

1. Data is received from the connected USB peripheral device and is displayed on the LCD.
    a) When a USB mouse is connected (Mouse mode), the displacement values of the X and Y axes are shown on the LCD. An LED is toggled by pressing the mouse buttons.
    b) When a USB keyboard is connected (Keyboard mode), show one character of the key input data from the USB keyboard report. Moreover, the *NumLock LED* is turned on when the device is in the configured state and the *NumLock LED* is turned off when the device is in the suspended state.
2. Suspends/Resume of USB device operations.
    a) The USB device is suspended and resumed alternately when SW3 on the RSK is pressed.
    b) Resume is executed when a remote wakeup signal is received from a USB device.

Switch input operation is described in Table 4-2.

Table 4-2 User switch input operation

Switch Function	Description	Switch Number
Data transfer start	Start ongoing requests for report reception.	SW2
State change	Change the following USB state. In data reception wait state (resumed), go to Suspend state. In Suspend state, go to data reception wait state (Resume).	SW3

4.3 File Configuration List

4.3.1 Folder Structure

The folder structure of the files supplied with the device class is shown below.

The source codes dependent on each MCU and evaluation board are stored in each hardware resource folder (***devicename****src**HwResource*).

```
workspace
  ＋[RL78 / R8C]
    ＋[ CCRL / CS+ / IAR / e2 studio / HEW ]
      ＋[ RL78G1C / R8C3MK / R8C34K ]
          ＋ HOST                                   Build result
          ＋ src
            ＋──── HIDFW [Human Interface Device Class driver ] See Table 4-3
            │        ＋──── inc                     Common header file of HID driver
            │        ＋──── src                     HID driver
            ＋────SmplMain [ Sample Application ]
            │        ＋──── APL                     Report display application
            ＋────USBSTDFW [Common USB code that is used by all USB firmware ]
            │        ＋──── inc                     Common header file of USB driver
            │        ＋──── src                     USB driver
            ＋──── HwResource [Hardware access layer; to initialize the MCU ]
                       ＋──── inc                   Common header file of hardware resource
                       ＋────src                    Hardware resource
```

[Note]

a. The project for CA78K0R compiler is stored under the CS+ folder.

b. The project for KPIT GNU compiler is stored under the e^2 studio folder.

c. Refer to **10 Using the e2 studio project with CS+** section when using CC-RL compiler on CS+.

4.3.2 File Structure

Table 4-3 shows the file structure provided in the HHID.

Table 4-3 File Structure

Folder	File Name	Description	Notes
HIDFW/inc	r_usb_class_usrcfg.h	USB host HID user definition	
HIDFW/inc	r_usb_hhid_define.h	HHID type definitions and macro definitions	
HIDFW/inc	r_usb_hhid_api.h	HHID prototype, external reference	
HIDFW/src	r_usb_hhid_api.c	HHID API functions	
HIDFW/src	r_usb_hhid_driver.c	HHID driver functions	
SmplMain	main.c	Main loop function	
SmplMain/APL	r_usb_hhid_apl.c	Sample application program	

4.4 System Resources

4.4.1 System Resource Definitions

Table 4-4 lists the Task ID and the task priority definitions used to register HHID in the scheduler. These are defined in the *r_usb_ckernelid.h* header file.

See 1.4 for why tasks are used.

Table 4-4 List of Scheduler Registration IDs

Scheduler registration task	Description	Notes
USB_HHID_TSK	**HHID** (R_usb_hhid_task) Task ID: USB_HHID_TSK Task priority: 2	
USB_HCD_TSK	**HCD** (R_usb_hstd_HcdTask) Task ID: USB_HCD_TSK Task priority: 0	
USB_MGR_TSK	**MGR** (R_usb_hstd_MgrTask) Task ID: USB_MGR_TSK Task priority: 1	
Mailbox ID / Default receive task	**Message description**	**Notes**
USB_HHID_MBX / USB_HHID_TSK	HHID -> HHID / APL -> HHID mailbox ID	
USB_HCD_MBX / USB_HCD_TSK	HCD task mailbox ID	
USB_MGR_MBX / USB_MGR_TSK	MGR task mailbox ID	

5. Host HID Sample Application Program (APL)

The host demo application performs display of received USB data when connected to a HID peripheral device. The HHID application complies with the USB Human Interface Device Class specifications. See Chapter 1.2 item 2 and 3.

5.1 Operating Environment

The Figure 5.1 and Figure 5.2 show a sample operating environment for the software.

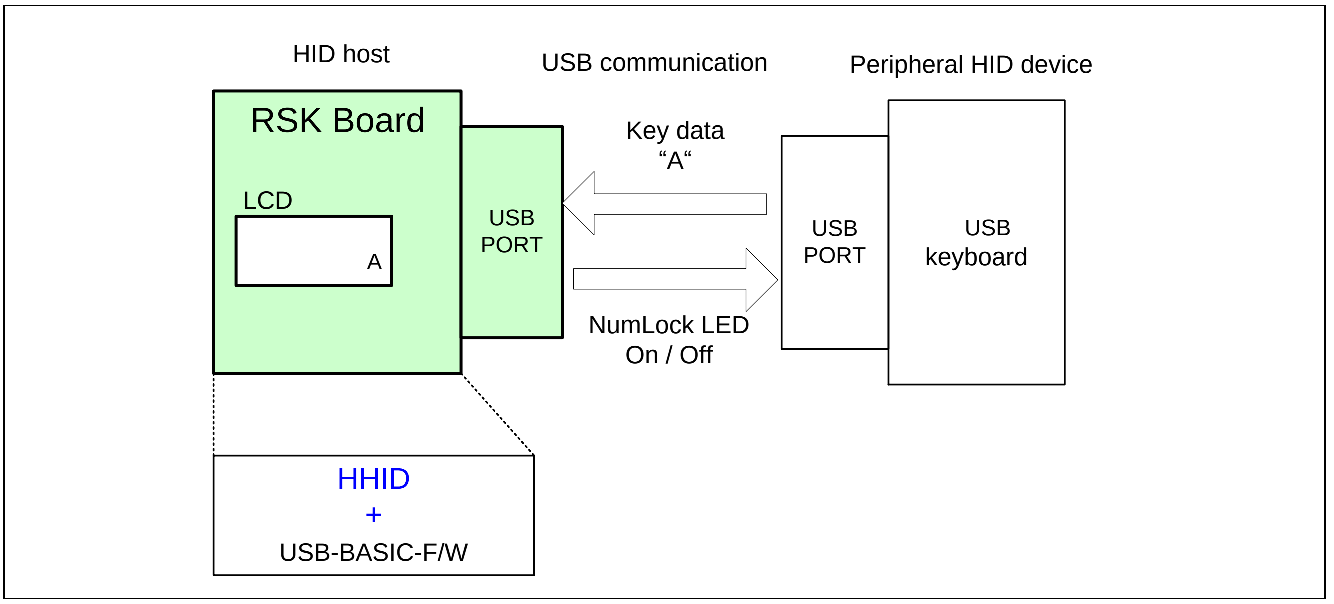

Figure 5.1 Example Operating Environment with a connected keyboard.

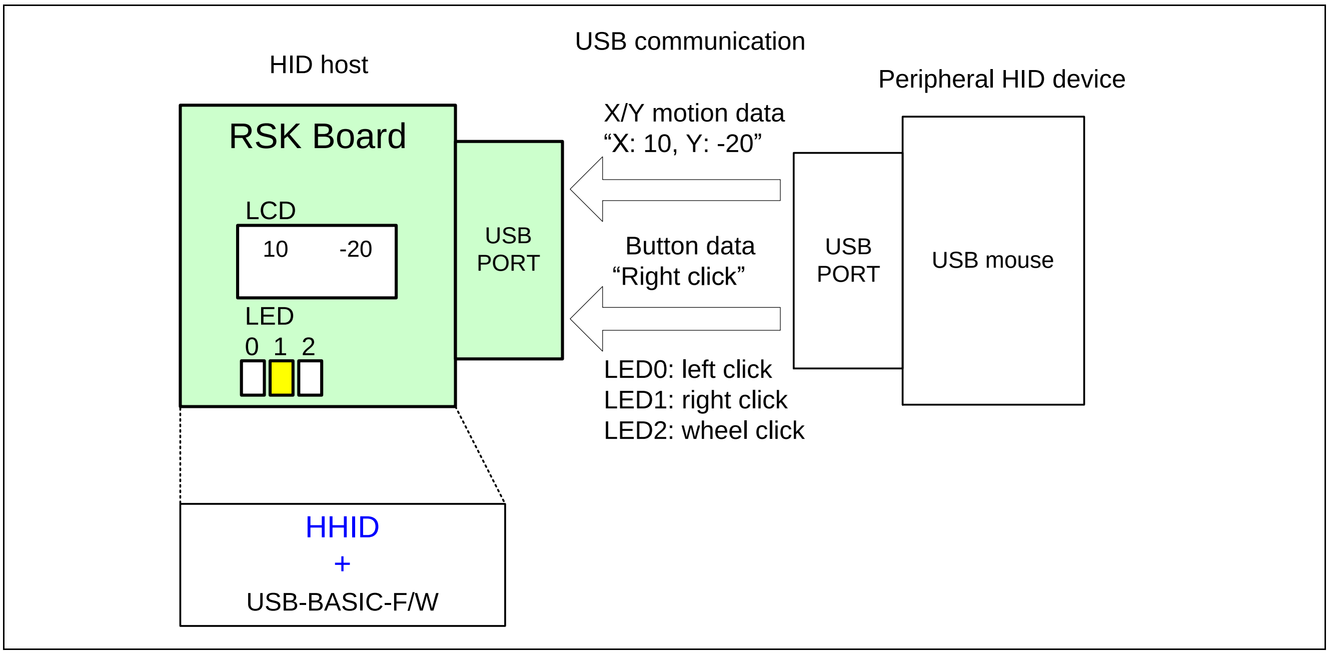

Figure 5.2 Example Operating Environment with a connected mouse.

5.1.1 Report reception

When the *USB_HHID_GET_REPORT_PIPE0* macro in the *r_usb_class_usrcfg.h* file is made active, report reception is made possible by the control transfer GET_REPORT request.

5.2 Description of Application Program Processing

The following lists application operation with respect to Figure 5.4, on page 12.

- HID peripheral device attachment. (Corresponding to Process No.0-1)

  Whether a connected device is a mouse or a keyboard is automaticallydetermined. The distinction between the two is done using *bInterfaceProtocol* of the Configuration descriptor (Refer to Table 5-1). Ther application program does not analyze the report descriptor.

Data Communication

- Start (Process 1-1):
  Communication with a USB device is started when SW2 is pressed (Refer to Table 4-2).
- Complete (Process 2-1):
  Communication with a USB device is completed when the callback function is generated from the HHID.

Operation

- During Data Communication (Process 3-1):
  Analyzes reports received from the peripheral device and displays them on LCD (Refer to Table 5-2).
- During Suspend state (Process 2-2):
  Data communication is terminated and the USB device is suspended when SW3 is pressed (Refer to Table 4-2).
- During Resume (Process 4-1):
  USB device is resumed and data communication restarts when SW3 is pressed (Refer to Table 4-2).

Table 5-1 Mode Switching

bInterfaceProtocol	Mode	Description
0x01	Keyboard Mode	Indicates that a keyboard device is connected
0x02	Mouse Mode	Indicates that a mouse device is connected
else	not possible to operate	Not recognized as an operable HID device connection.

Table 5-2 Operation During Data Communication

Mode	Description
Keyboard Mode	Display of received key data (key code to ASCII conversion)
Mouse Mode	Display of received coordinate data

5.3 Endpoint Specifications

The endpoints use by the HHID is shown in Table 5-3.

Table 5-3 Endpoint Specifications

Endpoint Number	Pipe Number	Transfer Method	Description
0	0	Control In/Out	Standard request, class request
Follows received Descriptor from attached device	6	Interrupt In	Data transfer from device to host

The Endpoint numbers are determined by the device's endpoint descriptors.

5.4 Allowed HID Peripherals

5.4.1 Supported Features

Full-Speed/Low-Speed keyboard s .

Three button mouse (FullSpeed/LowSpeed).

5.4.2 Non-supported Features

Devices with built-in HUB, or composite devices.

5.5 List of APL Functions

Table 5-4 lists the functions of the sample application.

Table 5-4 List of Functions of Sample Application

Function Name	Description
main	Main loop processing.
usb_hsmpl_main_init	System initialization. Task start up processing for Host USB.
usb_hhid_MainTask	Sample application main processing.
usb_hapl_registration	HHID driver registration.
usb_hhid_class_check	Check that connected device is a HID.
usb_hsmpl_device_state	Application status change callback function.
usb_hhid_smpl_data_trans_result	Data transfer complete processing.
usb_hhid_smpl_mse_data	Mouse data reception processing.
usb_hhid_smpl_val_to_str	1-byte numeric data string conversion processing.
usb_hhid_smpl_kbd_data	Keyboard data reception processing.
usb_hhid_smp_status_set	Sample application mode setting processing.
usb_hhid_smpl_get_hid_descriptor	HID descriptor processing.(not used).
usb_hhid_smpl_get_report_descriptor	Report descriptor getting processing.(not used).
usb_hhid_smpl_get_physical_descriptor	Physical descriptor getting processing.(not used).
usb_hhid_smpl_kbd_led_ctl	Keyboard LED ON/OFF control.
usb_hhid_smpl_set_report	SET REPORT request processing.(not used).
usb_hhid_smpl_get_report	GET REPORT request processing.(not used).
usb_hhid_smpl_set_idle	SET IDLE request processing.(not used).
usb_hhid_smpl_get_idle	GET IDLE request processing.(not used).
usb_hhid_smpl_set_protocol	SET PROTOCOL request processing.(not used).
usb_hhid_smpl_get_protocol	GET PROTOCOL request processing.(not used).
usb_hsmpl_class_result	HID class request callback function.
usb_hhid_smpl_get_report_result	GET REPORT request callback function.
usb_hhid_smpl_kbd_led_ctl_result	SET REPORT request callback function.

5.6 Host Application Task Sequence

The following explains how the LCD display is updated, and state transition controls.

5.6.1 Displayed Information

The application displays the USB device connection state and the content of reports received on the LCD.

When a keyboard is connected, the character of the last key pressed on the keyboard is displayed.

When a mouse is connected, the X/Y motion data is displayed. Values between -128 to 127 are displayed (right justified).

If the content of a received report is NULL (no key press on the keyboard or no XY motion from the mouse), the display on the LCD is not updated. The LCD display state transition is shown in Figure 5.3.

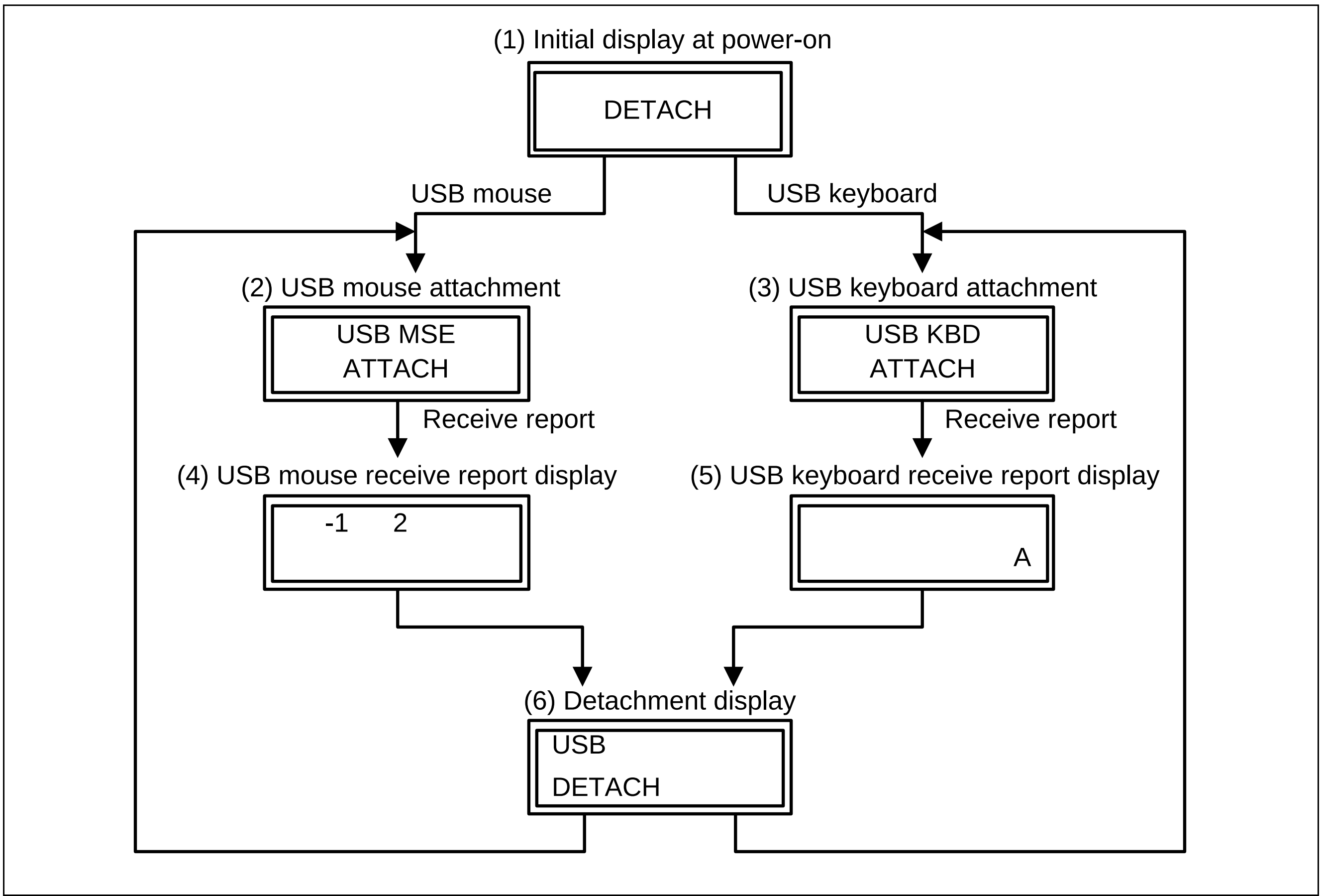

Figure 5.3 The Transition of the Display State on the LCD

5.6.2 State Transitions

Figure 5.4 shows the application state transition. Each block is a program "state".

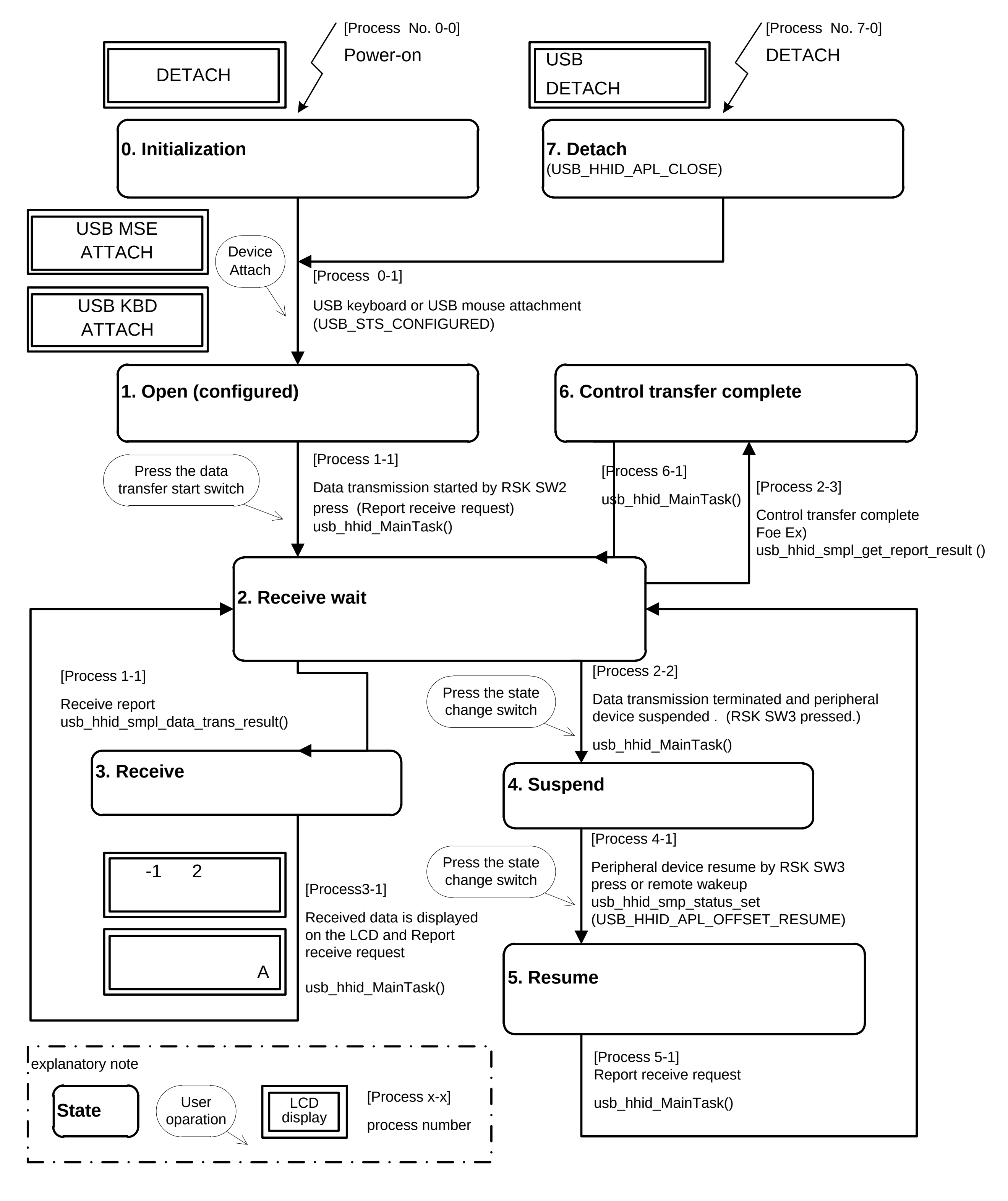

Figure 5.4 Application State Transitions

5.7 SW Processing Flow Graphs

The following shows the application task processing flow overview.

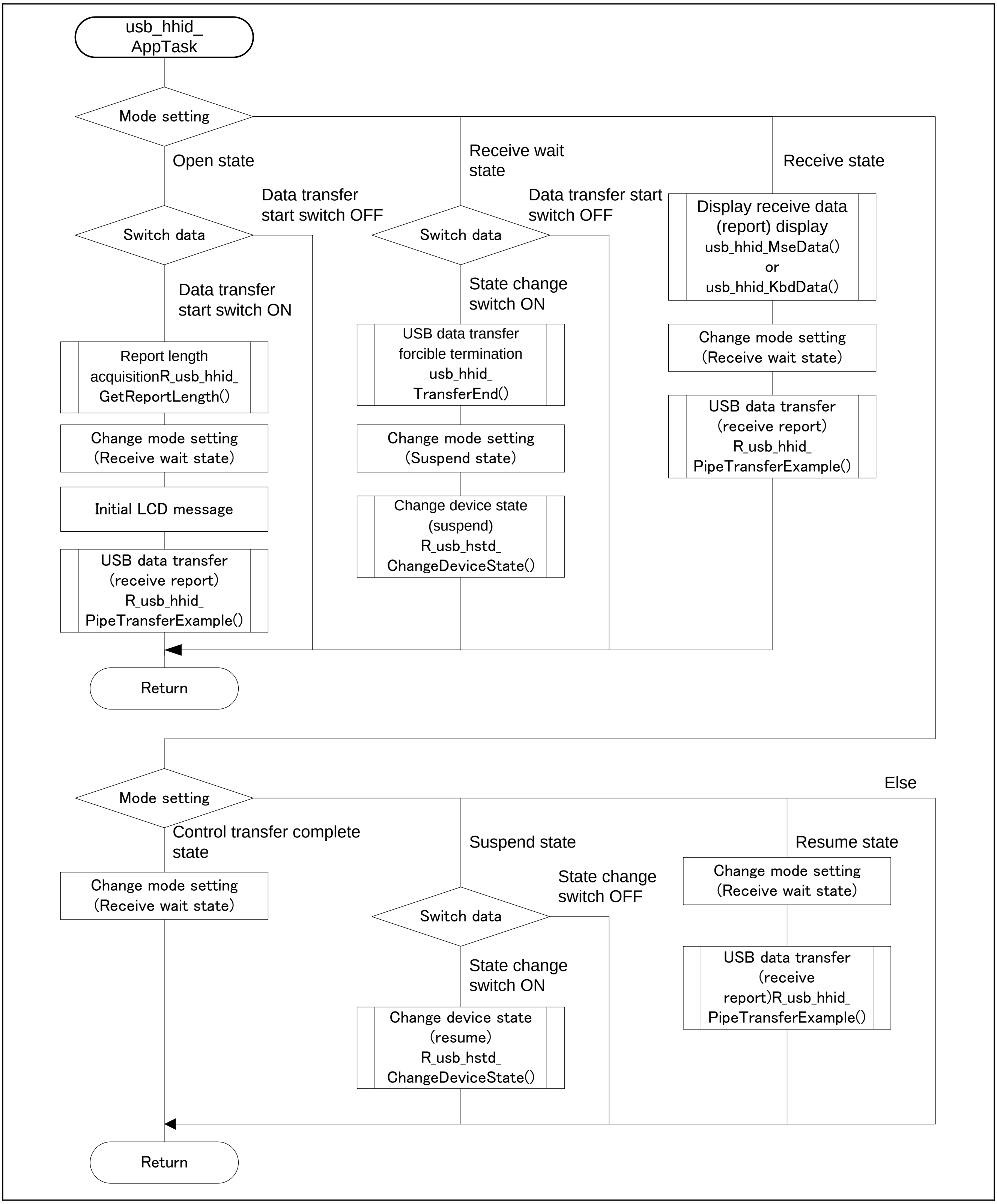

Figure 5.5 Application Task Processing Flow Overview

5.8 Sequences charts APL-HHID-HCD

The operation sequence of the sample application program is described below.

5.8.1 Startup to HID Device Attachment

The sequence from sample application program startup through completion of enumeration, application task startup, and completion of pipe control register setting is illustrated in Figure 5.6

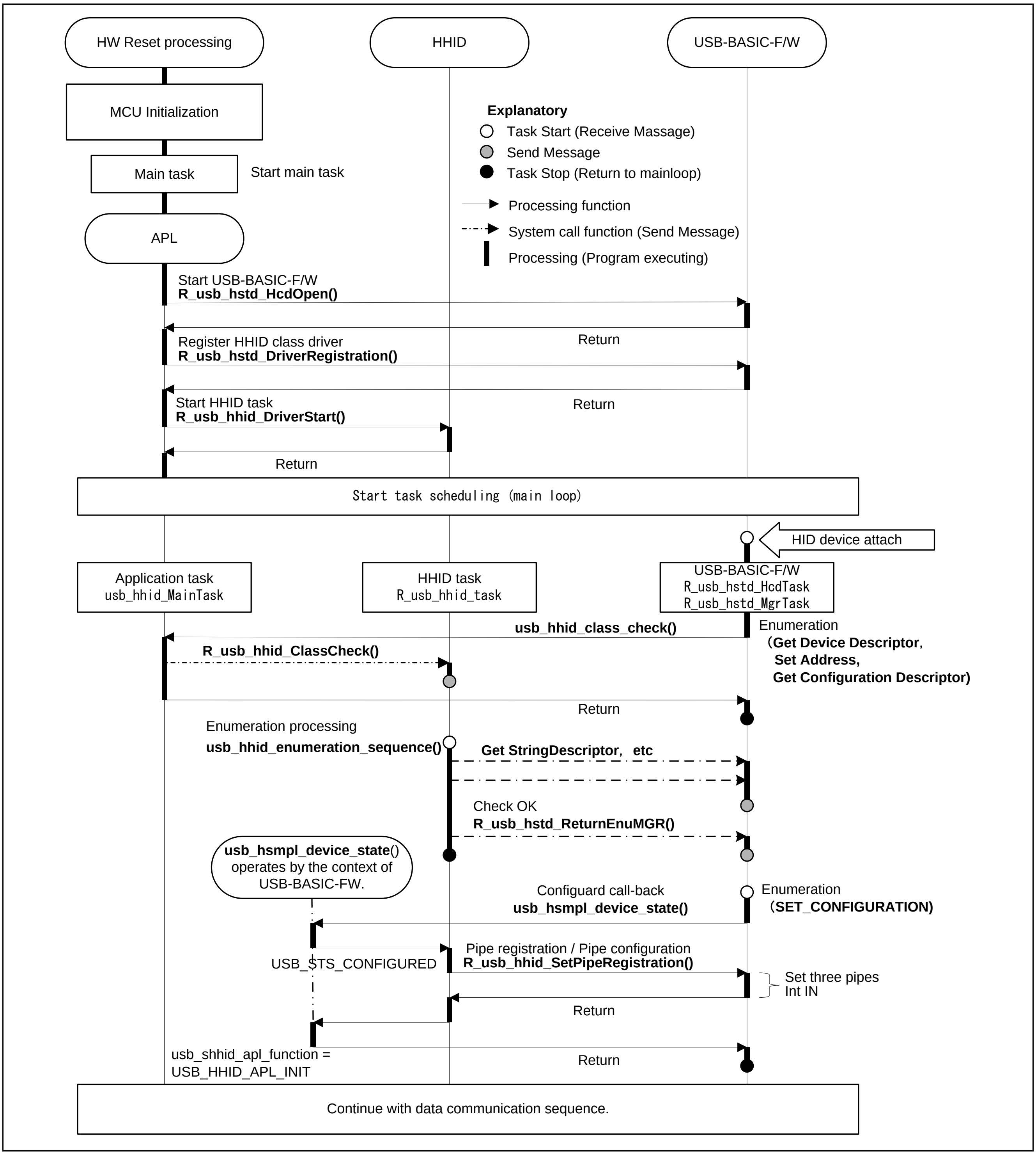

Figure 5.6 Startup to HID Device Attachment Sequence

5.8.2 Data Communication

Figure 5.7 and Figure 5.8 show the data transfer sequence that is connected by the keyboard device. The case where the report is received by the interrupt transfer is Figure 5.7. The case where the report is received by the control transfer is Figure 5.8.

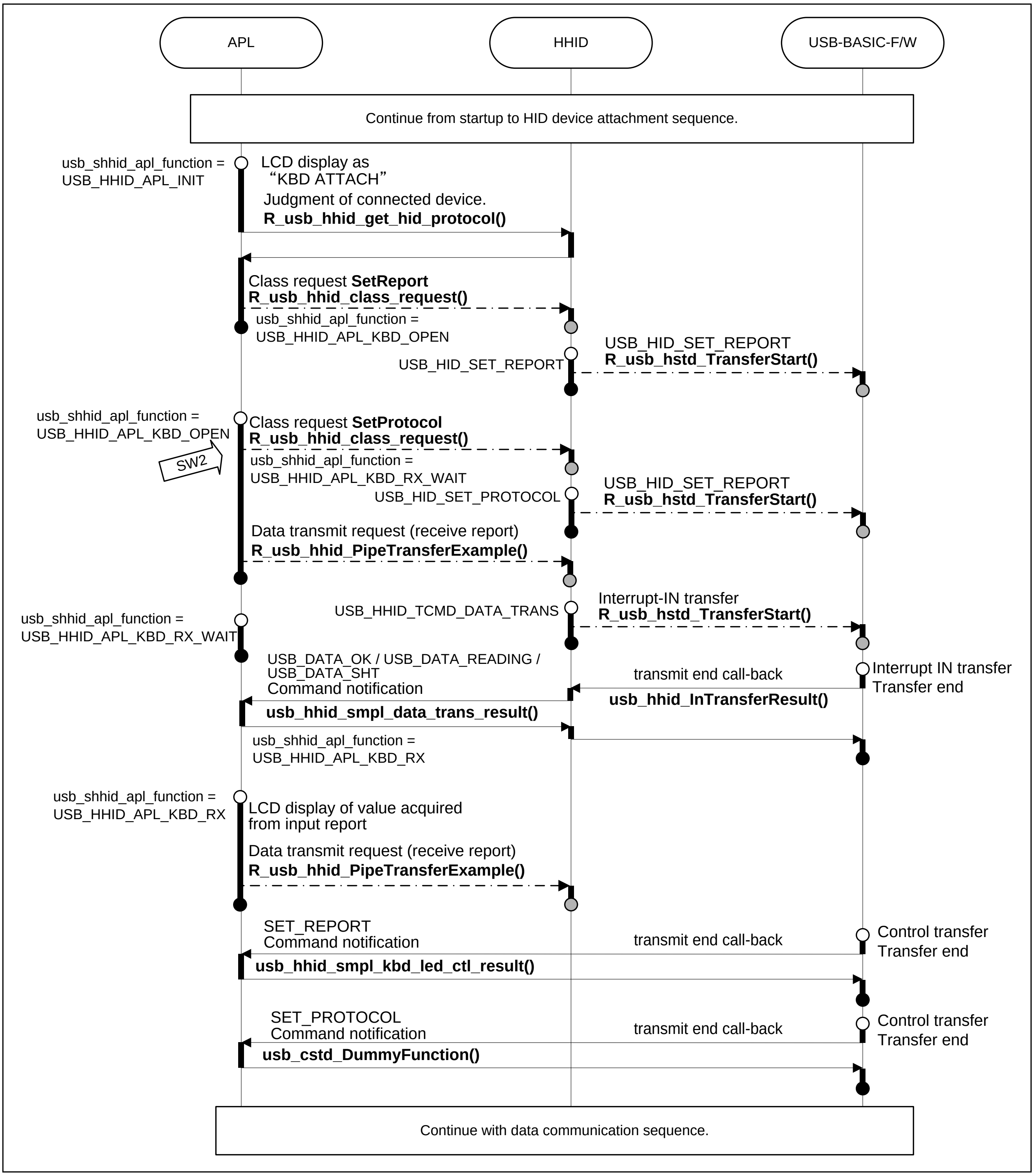

Figure 5.7 Interrupt-IN Communication Sequence by KBD

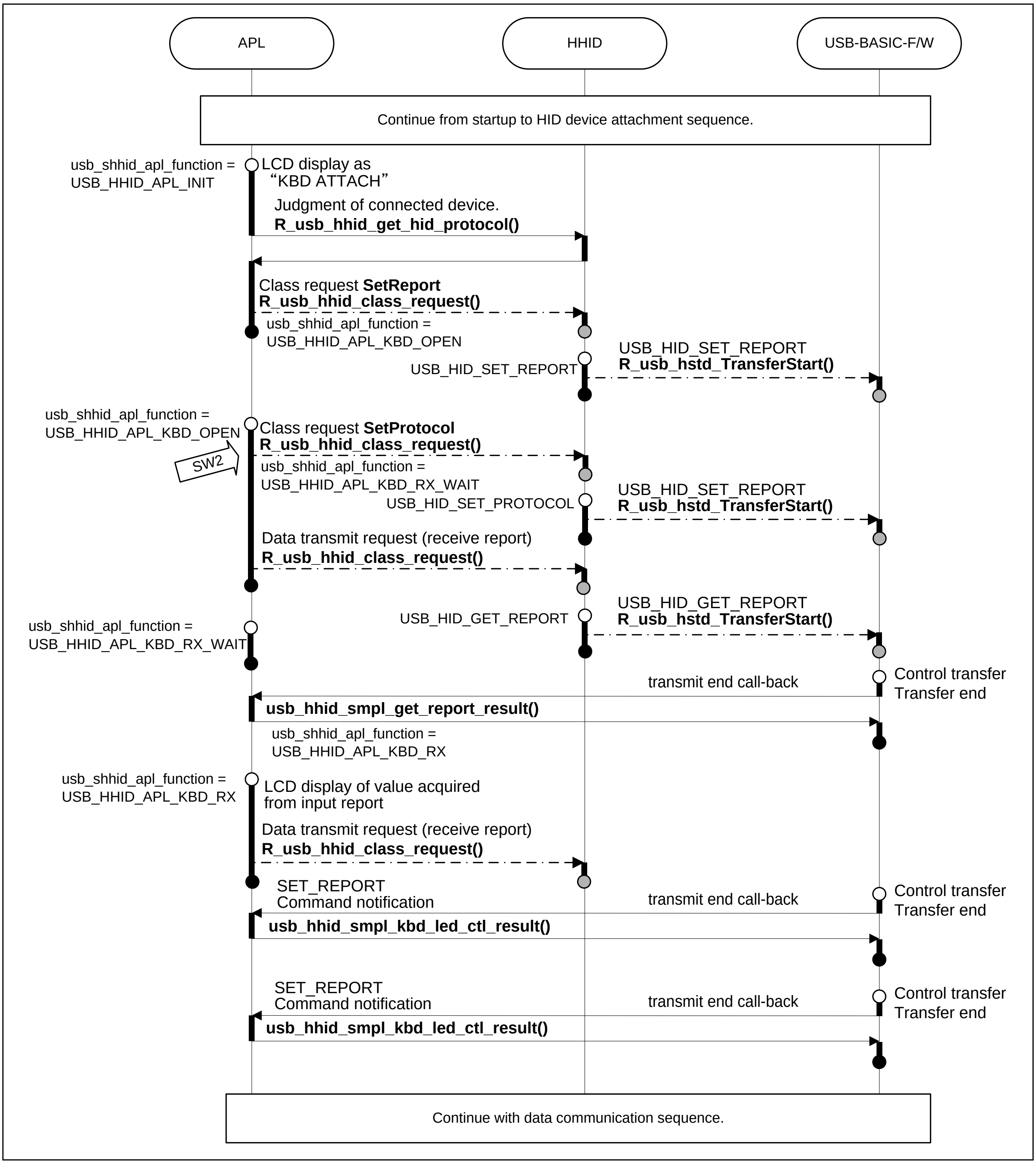

Figure 5.8 Control transfer Communication Sequence by KBD

Figure 5.9 and Figure 5.10 show the data transfer sequence that is connected by the mouse device. The case where the report is received by the interrupt transfer is Figure 5.9. The case where the report is received by the control transfer is Figure 5.10.

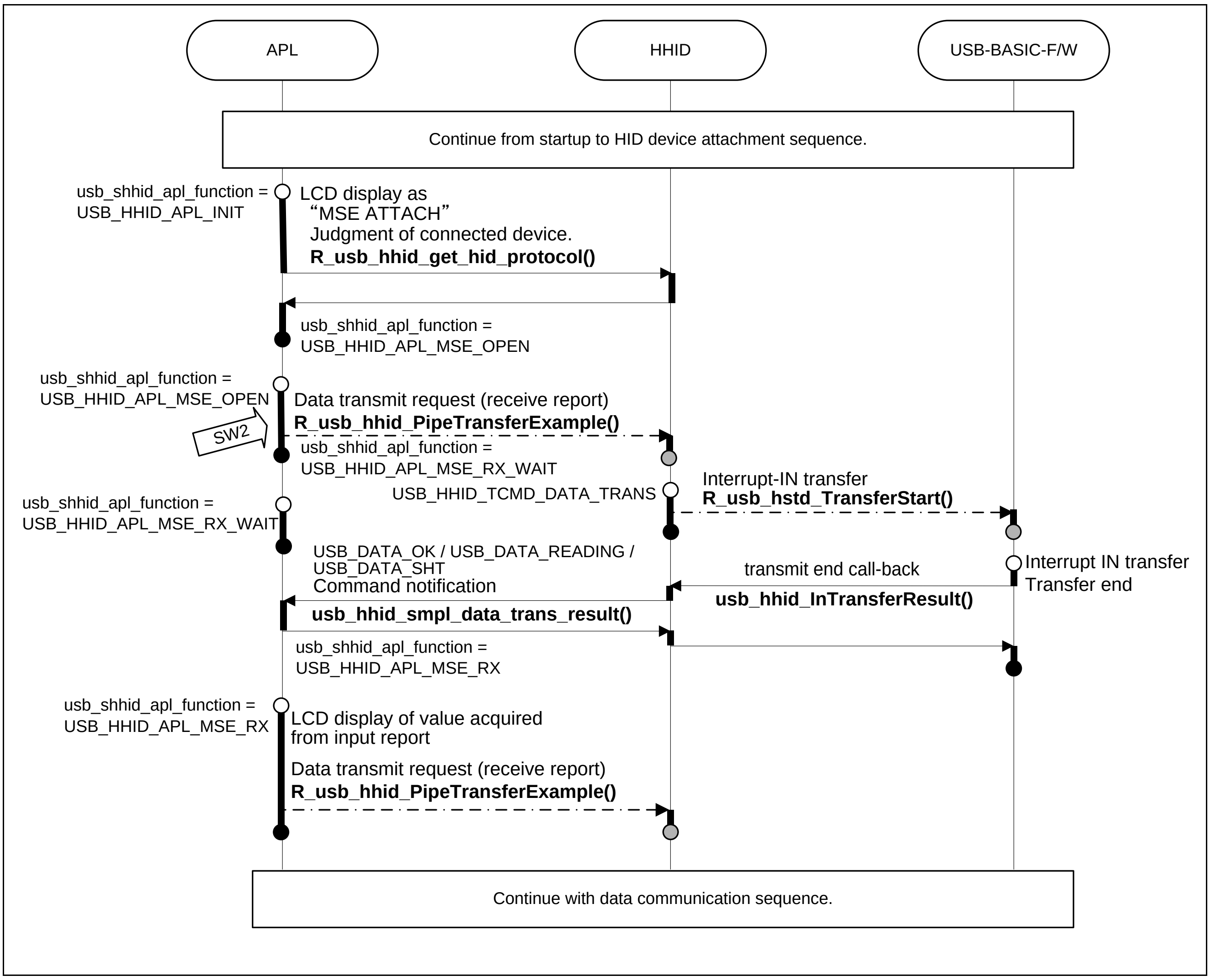

Figure 5.9 Interrupt-IN Communication Sequence by MSE

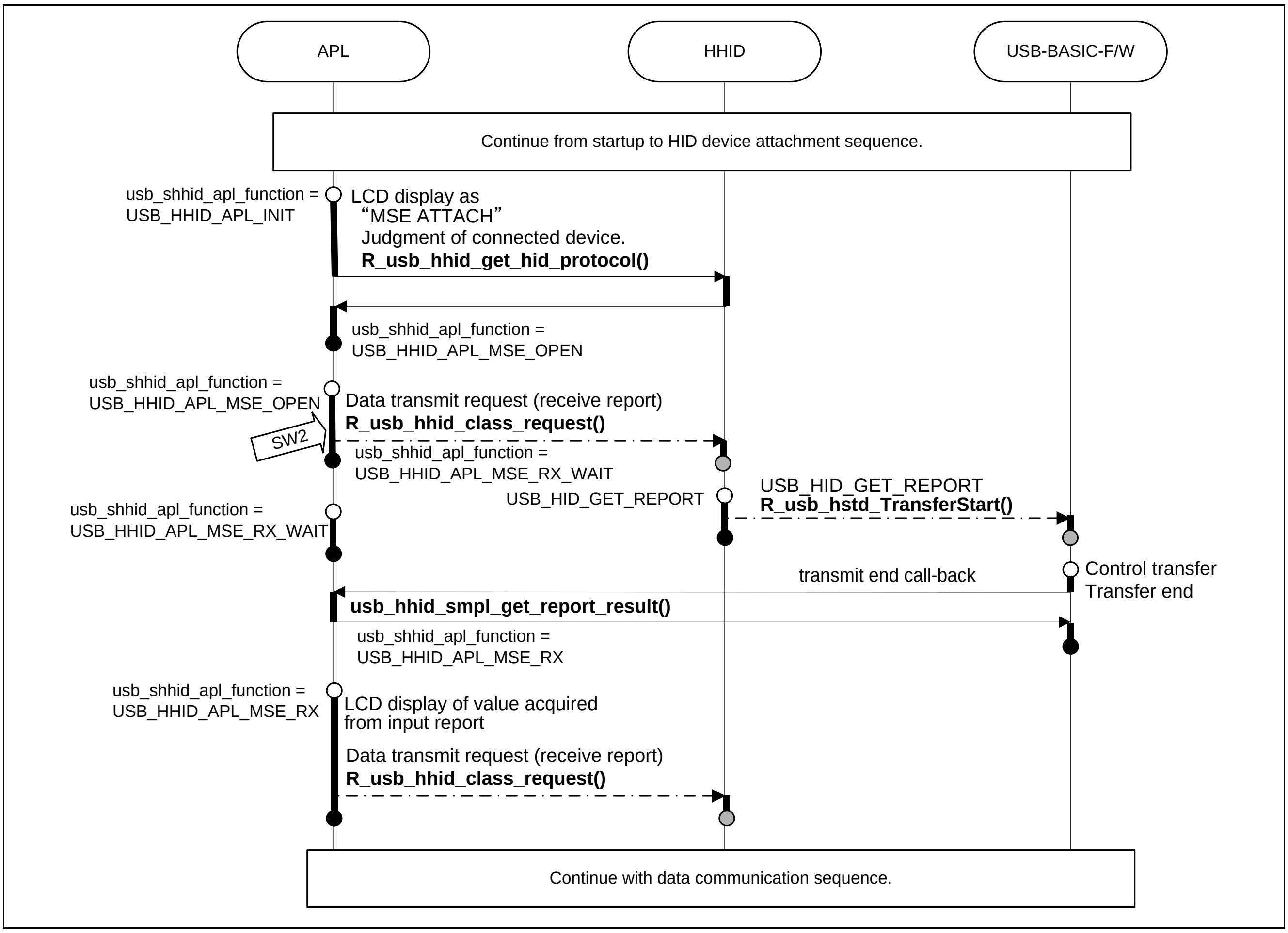

Figure 5.10 Control transfer Communication Sequence by MSE

5.8.3 HID Device Detach

The sequence when the HID device is detached is illustrated in Figure 5.11.

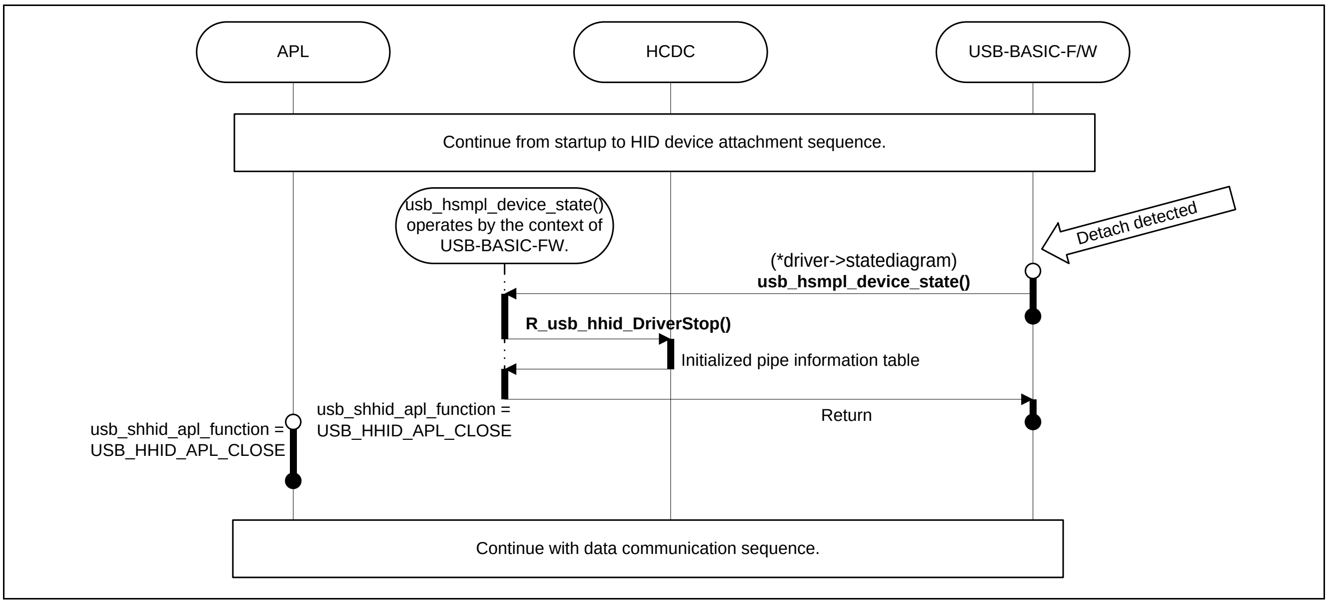

Figure 5.11 Device Detach Sequence

5.8.4 HID Device Suspended, Resumed

Figure 5.12 shows the suspend sequence. Figure 5.13 shows the resume sequence.

.

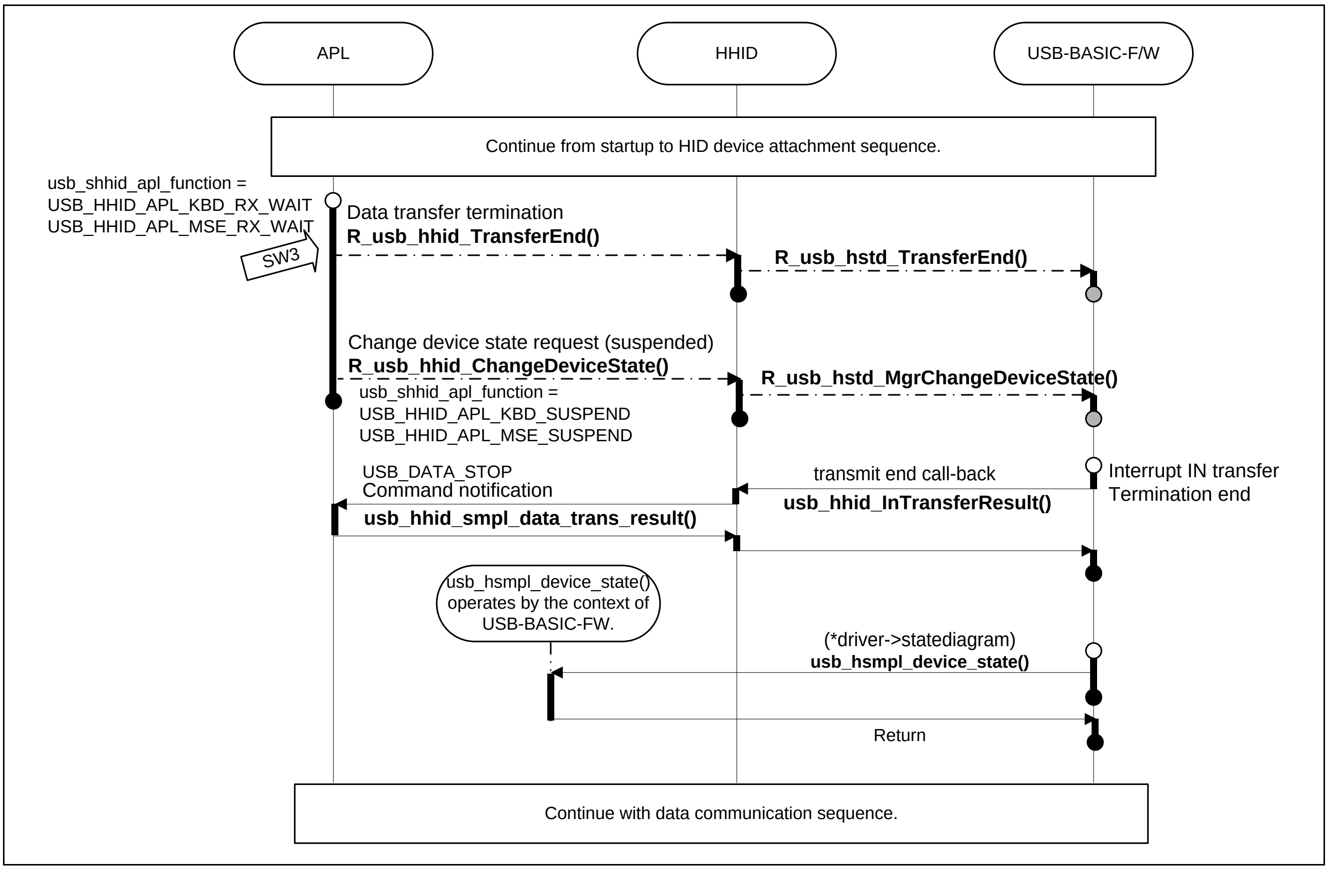

Figure 5.12 HID Device Suspend Sequence

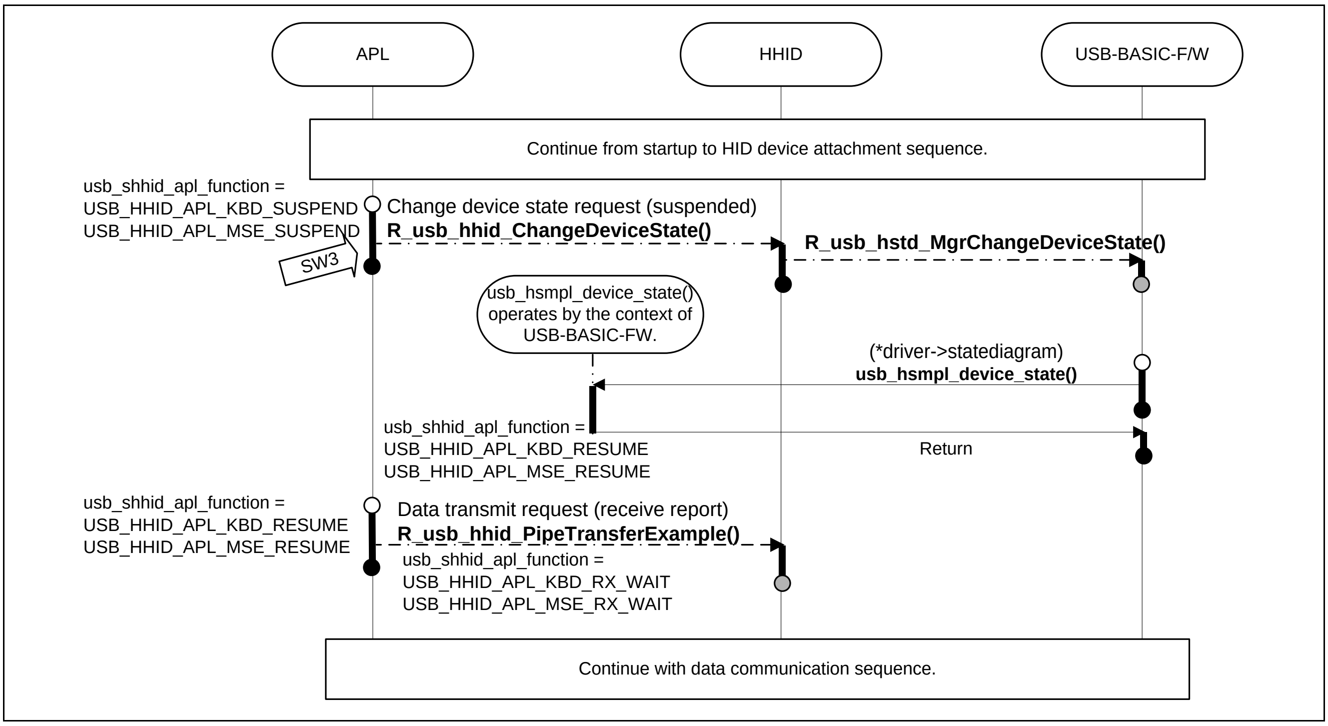

Figure 5.13 HID Device Resume Sequence

6. Human Interface Device Class (HID)

This software conforms to the Human Interface Device Class specification, as specified in the document listed in Chapter 1.2. The HID class consists primarily of devices that are used by humans to control the operation of computer input devices. Typical examples of HID class devices include:

- Keyboards and pointing devices - for example: standard mouse devices, trackballs, and joysticks.
- Front-panel controls - for example: knobs, switches, buttons, and sliders.
- Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices - for example: data gloves, throttles, steering wheels, and rudder pedals.

6.1 Basic Functions

The main functions are as follows.

1. Verify that connected devices are of type HID.
2. Inquire about the capabilities and state of a device.
3. Set the state of output and feature items.
4. Contro the transfer of data from the HID peripheral device.

6.2 HID Class Requests (Host to Device)

The software supports the following HID class requests.

Table 6-1 HID Requests

Request	Code	Description	Support
Get_Report	0x01	Receives a report from the HID device	Yes
Set_Report	0x09	Sends a report to the HID device	Yes
Get_Idle	0x02	Receives a duration (time) from the HID device	No
Set_Idle	0x0A	Sends a duration (time) to the HID device	No
Get_Protocol	0x03	Reads a protocol from the HID device	No
Set_Protocol	0x0B	Sends a protocol to the HID device	No
Get_Report_Descriptor	Standard	Transmit a report descriptor	Yes
Get_Hid_Descriptor	Standard	Transmit a HID descriptor	Yes

For details concerning the Requests, refer to Chapter 7 in “USB Device Class Definitions for Human Interface Devices”, Revision 1.1

7. USB Host Human Interface Device Class Driver (HHID)

7.1 Basic Functions

This software conforms to the Human Interface Device class specification. See Chapter 1.2 item 2 and 3.

The main functions of HHID are to:

1. Send class requests to the HID peripheral
2. Transfer data from the HID peripheral

7.2 HHID Task Description

This task receives messages in mailbox USB_HHID_MBX and performs processing according to the type of message. Table 7-1 shows processing according to message type.

Table 7-1 Processing according to Received HHID Message Type

Message	Processing	Message Source
USB_HHID_TCMD_OPEN	Gets the string descriptor and sets the pipe according the enumeration sequence.	R_usb_hhid_ClassCheck(). USB-BASIC-F/W and HHID check the connected device via this callback function during the enumeration.
USB_HHID_TCMD_DATA_TRANS	Start Interrupt-IN transfer. Notifies the application when the data transfer is completed.	R_usb_hhid_PipeTransferExample(). When Interrupt-IN transfer is completed this API function is executed.
USB_HHID_TCMD_CLASS_REQ	The HID class request is issued according to the demand of the application program shown by the argument. Notifies the application when the control transfer is completed.	R_usb_hhid_class_request(). This API function is called from the sample function which issues the class request.

7.3 Target Peripheral List (TPL)

A host class driver is not required to support operation of all USB peripherals of the class. It is up to the manufacturer of the host to determine what peripherals to support and provide a list of those peripherals. This is called the “Target Peripheral List (TPL)”.

TPL is composed of an array of supported VID(s) and PID(s). To not check VID (/PID), specify USB_NOVENDOR (/USB_NOPRODUCT). Refer to the *usb_gapl_devicetpl[]* array in the *r_usb_hhid_driver.c* file for the determination of TPL.

7.4 Structures

7.4.1 HHID Class API Function Structure

Table 7-2 describes the HID class request parameter structure.

Table 7-2 USB_HHID_CLASS_REQUEST_PARM_t Structure

Type	Member	Description
usb_addr_t	devadr	Device address.
uint8_t	bRequestCode	Class request code. Refer to the Table 7-3
void*	tranadr	Transfer data buffer.
usb_leng_t	tranlen	Transfer size.
uint16_t	duration	Response interval time rate to Interrupt transfer (4ms units).
uint8_t	set_protocol	Protocol value (Boot Protocol(=0)/Report Protocol(=1)).
uint8_t*	get_protocol	Protocol value stored address.
usb_cb_t	complete	Class request processing end call-back function.

7.4.2 HHID Class Request Code

Table 7-3 describes the code of the HID class requests.

Table 7-3 HHID Class Request code

Request Type	Definition Value	Support
Get_Descriptor(HID)	USB_HID_GET_HID_DESCRIPTOR	Yes
Get_Descriptor(Report)	USB_HID_GET_REPORT_DESCRIPTOR	Yes
Get_Descriptor(Physical)	USB_HID_GET_PHYSICAL_DESCRIPTOR	Yes
Set_Report	USB_HID_SET_REPORT	Yes
Get_Report	USB_HID_GET_REPORT	Yes
Set_Idle	USB_HID_SET_IDLE	No
Get_Idle	USB_HID_GET_IDLE	No
Set_Protocol	USB_HID_SET_PROTOCOL	No
Get_Protocol	USB_HID_GET_PROTOCOL	No

7.4.3 HID-Report Format

(1). Receive Report Format

Table 7-4 shows the receive report format used for notifications from the HID device.
Reports are received in Interrupt-IN transfers or class request *GetReport*.

Table 7-4 Receive report format

Offset / Application	Keyboard Mode	Mouse Mode
Data length	8 Bytes	3 Bytes
0 (Top Byte)	Modifier keys	b0: Button 1 b1: Button 2 b2-7: Reserved
+1	Reserved	X displacement
+2	Keycode 1	Y displacement
+3	Keycode 2	-
+4	Keycode 3	-
+5	Keycode 4	-
+6	Keycode 5	-
+7	Keycode 6	-

(2). Transmit Report Format

Table 7-5 shows the format of the transmit report sent to the HID device.
Reports are sent in the class request *SetReport*.

Table 7-5 Transmit report format

Offset / Application	Keyboard Mode	Mouse Mode
Data length	1 Bytes	Non-support
0 (Top Byte)	b0: LED 0 (NumLock) b1: LED 1(CapsLock) b2: LED 2(ScrollLock) b3: LED 3(Compose) b4: LED 4(Kana)	-
+1 ~ +16	-	-

(3). Note

The report format used by HID devices for data communication is based on the report descriptor. This HID driver does not acquire or analyze the report descriptor; rather, the report format is determined by the interface protocol code. User modifications must conform to the HID class specifications.

7.5 List of HHID API Functions

The HHID API is shown in Table 7-6.

Table 7-6 List of HHID API Functions

Function	Description	Notes
R_usb_hhid_task	HHID task processing	
R_usb_hhid_ClassCheck	This function requests the HHID task to judge whether the connected device is a HID device.	
R_usb_hhid_DriverStart	Start driver task HHID	
R_usb_hhid_DriverStop	Stop driver task HHID	
R_usb_hhid_SetPipeRegistration	Set pipe information table.	
R_usb_hhid_PipeTransferExample	USB data transfer request.	
R_usb_hhid_TransferEnd	USB data transfer termination request .	
R_usb_hhid_class_request	Send HID class request.	
R_usb_hhid_DeviceInformation	Acquire the USB state of a connected device.	
R_usb_hhid_ChangeDeviceState	Request USB status change of a connected device.	
R_usb_hhid_GetReportLength	Get the report length.	
R_usb_hhid_get_hid_protocol	Get Interface protocol value.	

R_usb_hhid_task

The HHID task

Format

void R_usb_hhid_task(void)

Argument

— —

Return Value

— —

Description

The HHID task function.

The HHID task processes requests from the application, and the results are notified to the application.

Note

Please refer to USB Basic Mini Firmware application note about task loops.

Example

```
void usb_apl_task_switch(void)
{
  while( 1 )
  {

   if( USB_FLGSET == R_usb_cstd_Scheduler())     /* Scheduler */
   {
      R_usb_hstd_HcdTask();    /* HCD Task */
      R_usb_hstd_MgrTask();    /* MGR Task */
      usb_hhid_main_task();    /* HHID Application Task */
      R_usb_hhid_task();       /* HHID Task */

   }
   else
   {
   }
  }
}
```

R_usb_hhid_ClassCheck

Check connected device's descriptors

Format

void R_usb_hhid_ClassCheck (uint8_t **table)

Argument

**table	Address array of the device information table
	[0] : Address of Device Descriptor
	[1] : Address of Configuration Descriptor
	[2] : Address of global variable that mean the Device Address

Return Value

— —

Description

This function requests the HHID task to determine whether the connected device is a HID device by studying the received descroptors. Call this function when the USB-BASIC-F/W executes the *classcheck* callback.

The HHID task references the endpoint descriptor(s) of the peripheral's configuration descriptor, then edits the Pipe Information Table, *usb_ghmsc_TmpEpTbl[],* and checks the pipe information of the pipes to be used.

Note

—

Example

```
USB_STATIC void usb_hhid_class_check(uint8_t **table)
{
        R_usb_hhid_ClassCheck(table);
        usb_shhid_smpl_devaddr = (usb_addr_t)(*table[2]);
}
```

R_usb_hhid_DriverStart

Start HHID driver

Format

void R_usb_hhid_DriverStart(void)

Argument

— —

Return Value

— —

Description

The function starts the HHID driver task.

Note

—

Example

```
void usb_hstd_task_start( void )
{
  /* Target board initialize */
  usb_cpu_target_init();

  /* USB-IP initialized */
  R_usb_hstd_ChangeDeviceState(USB_DO_INITHWFUNCTION);

  /* HCD driver open & registratuion */
  R_usb_hstd_HcdOpen();              /* HCD task, MGR task open */
  usb_hhid_registration();           /* HHID driver registration */
  R_usb_hhid_DriverStart();          /* HHID Task Start */

  /* Scheduler initialized */
  R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);

}
```

R_usb_hhid_DriverStop

Stop HHID driver

Format

void R_usb_hhid_DriverStop (void)

Argument

— —

Return Value

— —

Description

The function stops the HHID driver task.

Note

—

Example

```
USB_STATIC void usb_hsmpl_device_state(uint16_t data, uint16_t state)
{
  switch( state )
  {
   case USB_STS_DETACH:
        usb_smpl_set_suspend_flag(USB_NO);
        usb_shhid_active = USB_NO;
        usb_shhid_apl_function = USB_HHID_APL_CLOSE;
        R_usb_hhid_DriverStop();
   break;
                .
                .
                .
}
```

R_usb_hhid_SetPipeRegistration

Pipe and Pipe Information Table setting

Format

void R_usb_hhid_SetPipeRegistration(usb_addr_t devadr)

Argument

devadr Device address

Return Value

— —

Description

This function updates the address field of the host's Pipe Information table. It thereby sets the hardware pipe to be used for HID communication.

Note

1. Refer to USB Basic Mini Firmware application note for information on the Pipe Information Table.
2. Please set another field in the Pipe Information Table *usb_ghmsc_TmpEpTbl[]* beforehand by referring to the endpoint descriptor.

Example

```
void  usb_smp_task( void )
{
    :
  R_usb_hhid_SetPipeRegistration (devadr);
    :
}
```

R_usb_hhid_PipeTransferExample

USB data transfer request

Format

usb_er_t R_usb_hhid_TransferExample(uint8_t *table, usb_leng_t size, usb_cb_t complete)

Argument

*table	Pointer to the data buffer area.
size	Transfer data size
complete	Process completion callback function

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

This function requests a data transfer of the USB-BASIC-F/W.

The data of argument “*size*” byte is received at the address shown in argument “*table*”.

When the data reception processing is complete (data reception of "*size*" byte or short packet reception), the callback function is called.

Note

1. The data transfer process results are obtained by the argument “*usb_utr_t* *” of the callback function.
2. Refer to USB Basic Mini Firmware application note for the Data Transfer structure *usb_utr_t*.

Example

```
usb_er_t usb_smp_task(void)
{
  uint8_t    data[64];                          /* Data buff */
  usb_lenguint16_t size = 64;                   /* Data size */
                          :
                          :
  R_usb_hhid_TransferExample(data, size,(usb_cb_t)usb_data_received);

}

/* Callback function */
void usb_data_received(usb_utr_t *mess)
{
  /* Describe the processing performed when the USB receive is completed. */
}
```

R_usb_hhid_TransferEnd

USB data transfer termination request

Format

usb_er_t R_usb_hhid_TransferEnd(void)

Argument

— —

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap (transfer end request for the pipe during transfer end.)

Description

This function requests the USB-BASIC-F/W to end a data transfer in progress.

The transfer end is notified using the callback function set when the data transfer is requested (*R_usb_hhid_PipeTransferExample*, *R_usb_hhid_class_request*). The remaining data length of transmission and reception, pipe control register value, and transfer status = USB_DATA_STOP are set using the argument of the callback function *(usb_utr_t*).

The control transfer or the interrupt transfer is stopped according to how the *USB_HHID_GET_REPORT_PIPE0* macro in the *r_usb_class_usrcfg.h* file is set:

- *USB_HHID_GET_REPORT_PIPE0* macro enabled: Stop the control transfer.
- *USB_HHID_GET_REPORT_PIPE0* macro is disabled: Stop the interrupt transfer.

Note

1. The data transmit process forced end result is obtained by the argument "*usb_utr_t* *" of the callback function
2. Refer to USB Basic Mini Firmware application note for the Data Transfer structure *usb_utr_t*.

Example

```
void  usb_smp_task(void)
{

  /* Transfer end request */
  err = R_usb_hhid_TransferEnd(USB_PIPE6, USB_DO_TRANSFER_STP);

  return err;
    :
}
```

R_usb_hhid_class_request

Send HID class request

Format

usb_er_t R_usb_hhid_class_request(USB_HHID_CLASS_REQUEST_PARM_t *pram)

Argument

*pram HID class request structure. Refer to Chapter 7.4 for the *USB_HHID_CLASS_REQUEST_PARM_t* argument structure.

Return Value

— Error code (USB_E_OK/USB_E_ERROR)

Description

The following HID class requests can be sent to the HHID driver.

Judges the request type by the structure member *bRequestCode* of argument **parm*.

1. Get_Descriptor(HID)
2. Get_Descriptor(Report)
3. Get_Descriptor(Physical)
4. Set_Report
5. Get_Report
6. Set_Idle
7. Get_Idle
8. Set_Protocol
9. Get_Protocol

Please refer to the sample application in *r_usb_hhid_apl.c* for details on how to use.

Note

1. The class request transmission result is obtained via the argument "*usb_utr_t* *" of the callback function.
2. Refer to USB Basic Mini Firmware application note for the Data Transfer structure *usb_utr_t*.

Example

```
void usb_hhid_smpl_set_report(uint16_t devadr, uint8_t *p_data, uint16_t
length, usb_cb_t complete)
{
  USB_HHID_CLASS_REQUEST_PARM_t   class_req;
  /* SET_REPORT */
  class_req.bRequestCode = USB_HID_SET_REPORT;
  class_req.devadr    = devadr;
  class_req.tranadr   = p_data;
  class_req.tranlen   = length;
  class_req.complete  = complete;
   R_usb_hhid_class_request(class_req);
}
```

R_usb_hhid_DeviceInformation

Obtain USB device state and other information

Format

void R_usb_hhid_DeviceInformation(uint16_t *deviceinfo)

Argument

*deviceinfo Table address to store the device information

Return Value

— —

Description

Obtain the connected USB device information. The following information will be stored to the address specified by the argument "**deviceinfo*":

[0]: Root port number (port 0: USB_0, port 1: USB_1)
[1]: USB state (unconnected: USB_STS_DETACH, enumerated: USB_STS_DEFAULT/USB_STS_ADDRESS, connected: USB_STS_CONFIGURED, suspended: USB_STS_SUSPEND)
[2]: Structure number (*g_usb_HcdDevInfo[g_usb_MgrDevAddr].config*)
[3]: Connection speed (FS: USB_FSCONNECT, LS: USB_LSCONNECT, unconnected: USB_NOCONNECT)

Notes

1. Provide an area of 4 words for the argument *deviceinfo.
2. This function is called when the device address is 0, the following information is returned.
   (1) When there is not a device during enumeration (device is not connected).
   table[0] = USB_NOPORT, table[1] = USB_STS_DETACH
   (2) When there is a device during enumeration.
   table[0] = Port number, table[1] = USB_STS_DEFAULT

Example

```
void  usb_smp_task(void)
{
  uint16_t  tbl[4];
    :
  /* Device information check */
  R_usb_hhid_DeviceInformation(tbl);
    :
}
```

R_usb_hhid_ChangeDeviceState

USB device state change request

Format

```
usb_er_t        R_usb_hhid_ChangeDeviceState (usb_strct_t msginfo,
                                              usb_strct_t keyword,
                                              usb_cb_info_t complete)
```

Arguments

msginfo	USB state to change into. States are listed below.
keyword	Content depends on *msginfo*. For example, it would be port number if the port is to be disabled.
complete	Callback function executed when the USB state changing ends.

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

Set the following value to argument *msginfo* and request to change the device state to the USB-BASIC-F/W.

- USB_DO_PORT_ENABLE / USB_DO_PORT_DISABLE
  Enable or disable a port specified by a keyword (on/off control of VBUS output).
- USB_DO_GLOBAL_SUSPEND
  Suspend a port specified by a keyword.
- USB_DO_GLOBAL_RESUME
  Resume a port specified by a keyword.
- USB_DO_CLEAR_STALL
  Cancel STALL of the device that uses a pipe specified by a keyword.

Notes

1. When a connection or disconnection is detected by the USB-BASIC-F/W, USB-BASIC-F/W automatically does enumeration or the detach sequence processing.
2. When changing the USB state using this function, the USB state transition callback of the driver structure registered using the API function *R_usb_hstd_DriverRegistration()* is not called.

Example

```
void  usb_smp_task(void)
{
  R_usb_hhid_ChangeDeviceState
   (USB_DO_GLOBAL_SUSPEND, USB_PORT0, usb_hsmpl_status_result);
}
```

R_usb_hhid_GetReportLength

Gets HID Report length

Format

uint16_t R_usb_hhid_GetReportLength(void)

Argument

— —

Return Value

— Max packet size

Description

This function gets the max packet size of the connected USB device.

Note

Example

```
void usb_smp_task( void )
{
  uint16_t   usb_smp_report_length;
     :
  usb_smp_report_length = R_usb_hhid_GetReportLength();
     :
}
```

R_usb_hhid_get_interfaceprotocol

Get interface protocol value

Format

uint8_t R_usb_hhid_get_interfaceprotocol(void)

Argument

— —

Return Value

— Protocol code of USB device（*bInterfaceProtocol*）

Description

This function gets the interface protocol value of the connected USB device.

Note

1. *bInterfaceProtocol* is included in Interface Descriptor.
2. The protocol code of the first HID class is sent as response for the multi interface device.

Example

```
void  usb_smp_task( void )
{
  uint8_t   protocol;
    :
  /* Gets the interface protocol value */
  protocol = R_usb_hhid_get_interfaceprotocol();
    :
}
```

8. Limitations

The following limitations apply to HHID.

1. Only one device can connect to HHID. Please do not connect two or more devices simultaneously.
2. The HID driver must analyze the report descriptor to determine the report format. This HHID driver determines the report format only from the interface protocol.
3. The structures contain members of different types. Depending on the compiler, this may cause address misalignment of structure members.

9. Setup for the e^2 studio project

(1). Start up e^2 studio.

* If starting up e^2 studio for the first time, the Workspace Launcher dialog box will appear first. Specify the folder which will store the project.

(2). Select [File] → [Import]; the import dialog box will appear.

(3). In the Import dialog box, select [Existing Projects into Workspace].

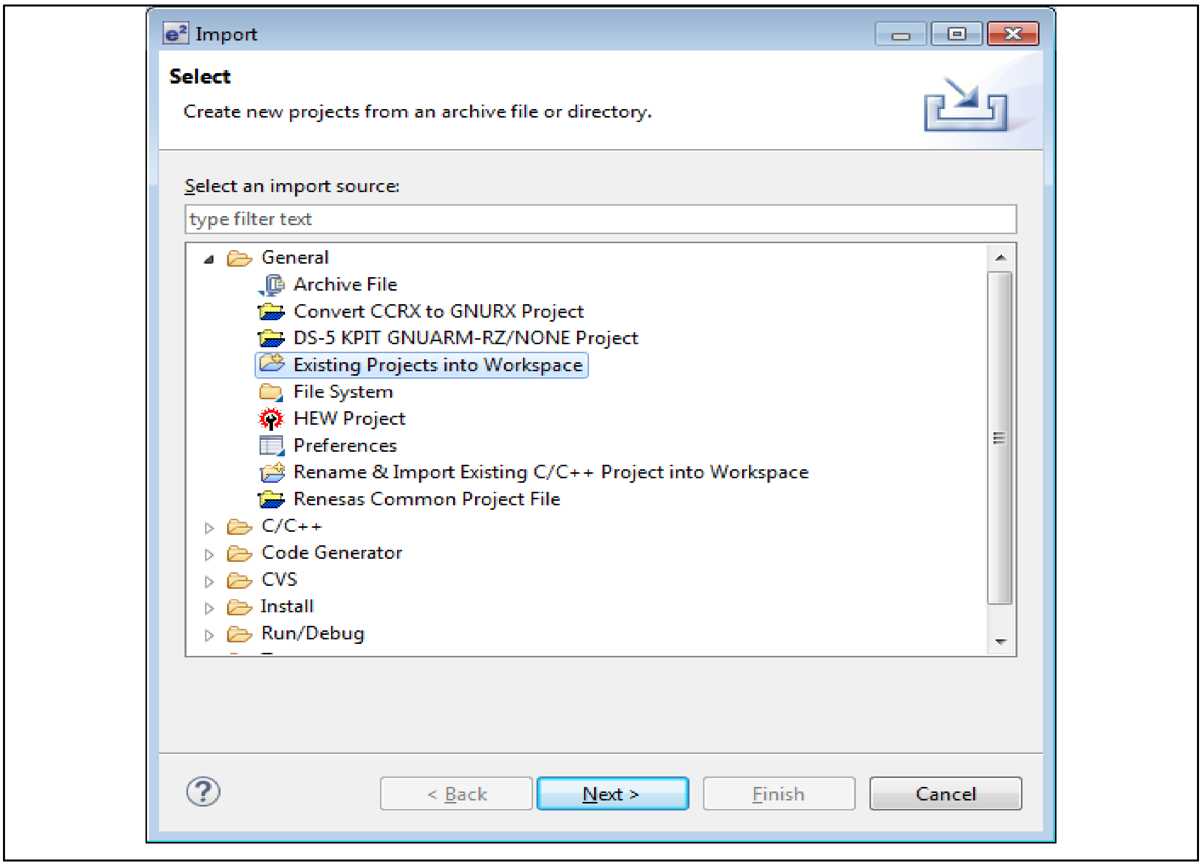

Figure 9-1 Select Import Source

(4). Press [Browse] for [Select root directory]. Select the folder in which [.cproject] (project file) is stored.

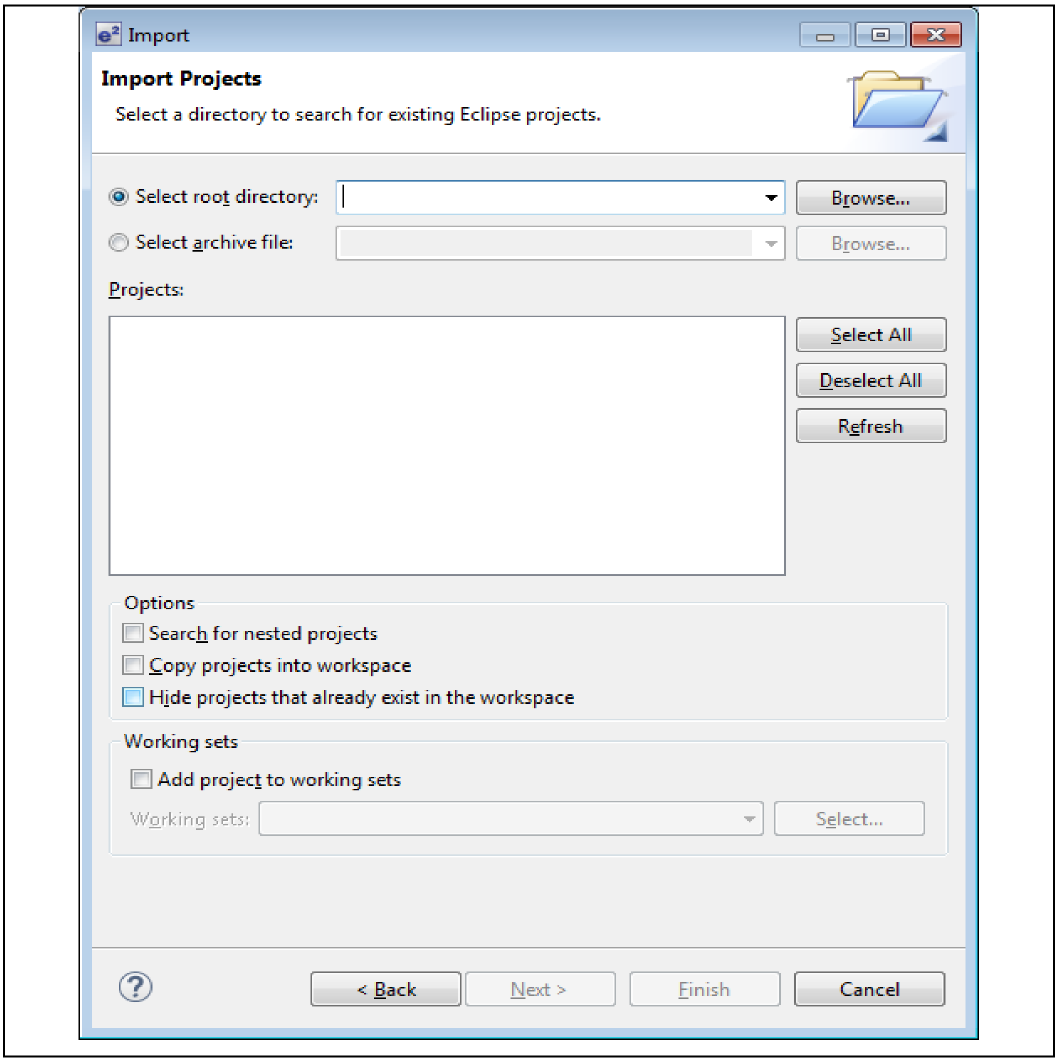

Figure 9-2 Project Import Dialog Box

(5). Click [Finish].

This completes the step for importing a project to the project workspace.

10. Using the e² studio project with CS+

This package contains a project only for e² studio. When you use this project with CS+, import the project to CS+ by following procedures.

Note:

The *rcpc* file is stored in "workspace\RL78\CCRL*devicename*" folder.

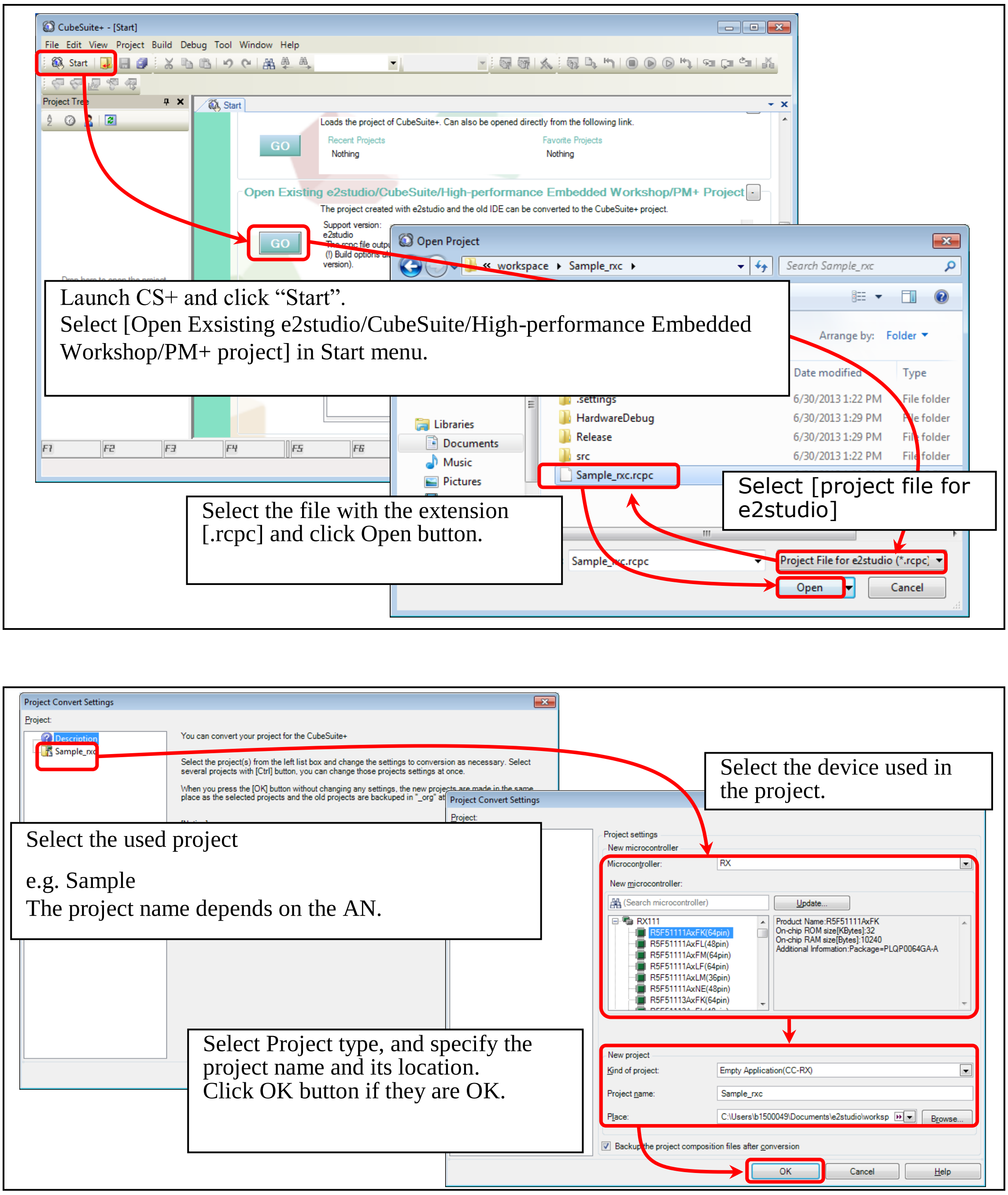

Figure 10-1 Using the e² studio project with CS+

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision Record

Rev.	Date	Description	
		Page	Summary
0.00	May. 12.11	—	First edition issued
2.00	Nov. 30.12	—	Revision of the document by firmware upgrade
2.10	Aug. 01. 13	—	RX111 is supported. Error is fixed.
2.11	Oct. 31. 13	—	1.4 Folder path fixed. 3.3.1 Folder Structure was corrected. Error is fixed.
2.12	Mar. 31. 14	—	R8C is supported. Error is fixed.
2.13	Mar. 16. 15	—	RX111 is deleted from Target Device
2.14	Jan. 18. 16	—	Supported Technical Update (Document No. TN-RL*-A055A/E)
2.15	Mar. 28. 16	—	CC-RL compiler is supported.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0