RZ/T1 Group

R01AN2636EJ0130
Rev.1.30
Dec 07, 2017

USB Host Human Interface Device Class Driver (HHID)

Introduction

This application note describes USB Host Human Interface Device Class Driver. This module performs hardware control of USB communication. It is referred to below as the USB-BASIC-F/W.

The sample program of this application note is created based on "RZ/T1 group Initial Settings Rev.1.30". Please refer to "RZ/T1 group Initial Settings application note (R01AN2554EJ0130)" about operating environment.

Target Device

RZ/T1 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

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. RZ/T1 Group User's Manual: Hardware (Document No.R01UH0483EJ0130)
5. RZ/T1 Group Initial Settings (Document No.R01AN2554EJ0130)
6. USB Host Basic Firmware (Document No.R01AN2633EJ0130)

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

Contents

1. Overview

The USB HHID, when used in combination with the USB-BASIC-F/W, operates as a USB host human interface device class driver.

This module supports the following functions.

- Data communication with a connected HID device (USB mouse, USB keyboard)
- Issuing of HID class requests to a connected HID device
- Connect multiple HID devices.

Limitations

The following limitations apply to the HHID.

The structures contain members of different types. (Depending on the compiler, this may cause address misalignment of structure members.)

The HID driver must analyze the report descriptor to determine the report format (This HID driver determines the report format from the interface protocol alone.)

HID devices that perform Interrupt OUT transfer are not supported.

Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

APL	:	Application program
HCD	:	Host control driver of USB-BASIC-FW
HDCD	:	Host device class driver (device driver and USB class driver)
HHID	:	Host human interface device
HID	:	Human interface device class
HUBCD	:	Hub class sample driver
MGR	:	Peripheral device state manager of HCD
Scheduler	:	Used to schedule functions, like a simplified OS.
Task	:	Processing unit
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB basic firmware for RZ/T1 Group

2. Software Configuration

Figure 2-1 shows the structure of the HHID-related modules. Table 2-1 lists the modules and an overview of each.

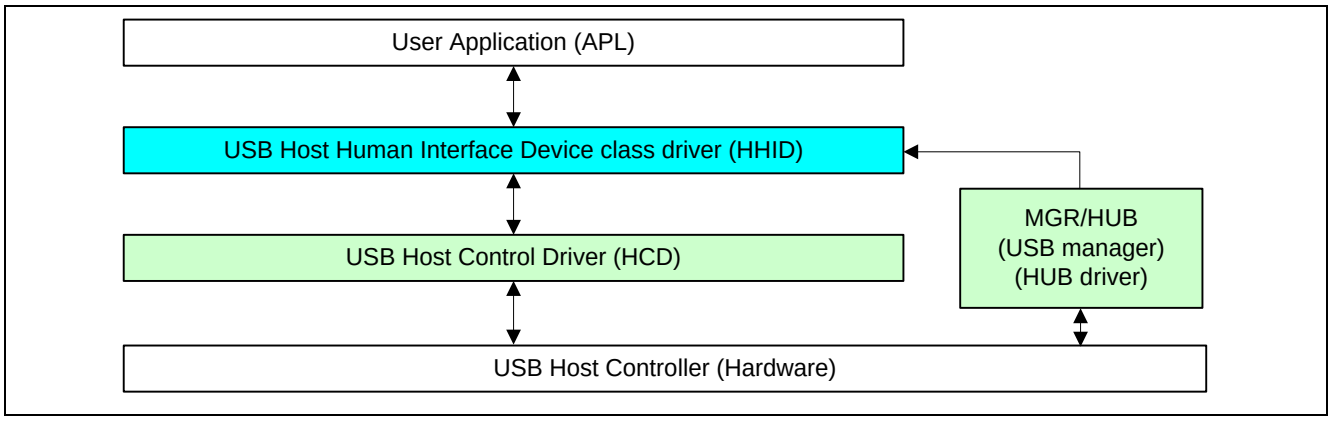


Figure 2-1 Software Configuration

Table 2-1 Module Function Descriptions

Module Name	Description
APL	User application program. (Please prepare for your system)
HHID	USB Host Human Interface Device Class Driver. HID device access. Requests HID requests command and the data transfer from APL to HCD.
MGR / HUB	USB Manager / HUB class driver. (USB-BASIC-F/W) Enumerates the connected devices and starts HHID Performs device state management.
HCD	USB host Hardware Control Driver. (USB-BASIC-F/W)

3. USB Human Interface Device Class Driver (HHID)

HHID conforms to the human interface device class specification.

3.1 Basic Functions

This software complies with the HID class specification. The main functions of the driver are as follows.

(1) HID device access

(2) Class request notifications to the HID device

(3) Data communication with the HID device

3.2 Class Requests

Table 3-1 shows the class requests supported by the HHID.

Table 3-1 HID Class Requests

Request	Code	Description
USB_SET_REPORT	0x09	Sends a report to the HID device
USB_SET_PROTOCOL	0x0B	Sends a protocol to the HID device

The class request data formats supported in this software are described below.

a). SetReport Request Format

Table 3-2 shows the SetReport request format.
Sends report data to the device in a control transfer.

Table 3-2 SetReport Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_REPORT (0x09)	ReportType & ReportID	Interface	ReportLength	Report

b). SetProtocol Request Format

Table 3-3 shows the SetProtocol request format.
Sets protocol (boot protocol or report protocol).

Table 3-3 SetProtocol Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_PROTOCOL (0x0B)	0(BootProtocol) / 1(ReportProtocol)	Interface	0x0000	Not applicable

3.2.1 Report Formats

The report format supported in this software are described below.

(1). Receive Report Format

Table 3-4 shows the receive report format used for notifications from the HID device.
Reports are received in interrupt IN transfers or class request GetReport.

Table 3-4 Receive Report Format

Offset (Data length)	Keyboard Mode (8 Bytes)	Mouse Mode (3 Bytes)
0 (Top Byte)	Modifier keys	b0: Button 1 b1: Button 2 b2: Button 3 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 3-5 shows the format of the transmit report sent to the HID device.
Reports are sent in the class request SetReport.

Table 3-5 Transmit Report Format

Offset (Data length)	Keyboard (1 Byte)	Mouse (Not supported)
0 (Top Byte)	b0: NumLock b1: CapsLock b2: ScrollLock b3: Compose b4: 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 get 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.

3.3 Scheduler settings

Table 3-6 shows Scheduler settings of HHID.

Table 3-6 Scheduler settings

Function	ID	Priority	Mailbox ID	Memory Pool ID	Desctiption
R_usb_hhid_task	USB_HHID_TSK	USB_PRI_3	USB_HHID_MBX	USB_HHID_MPL	HHID Task
R_usb_hub_task	USB_HUB_TSK	USB_PRI_3	USB_HUB_MBX	USB_HUB_MPL	HUB Task
R_usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	USB_MGR_MBX	USB_MGR_MPL	MGR Task
r_usb_hstd_HciTask	USB_HCI_TSK	USB_PRI_1	USB_HCI_MBX	USB_HCI_MPL	HCD Task

3.4 API

All API calls and their supporting interface definitions are located in r_usb_hhid_if.h.

Please modify r_usb_hhid_config.h when User sets the module configuration option.

Table 3-7 shows the option name and the setting value.

Table 3-7 Configuration options

Name	Default	Description
MAX_DEVICE_NUM	4	Max connect device number

Table 3-8 shows the HHID driver API.

Table 3-8 HHID API Function List

Function	Description
R_usb_hhid_task	HHID Task
R_usb_hhid_class_check	Descriptor checking process
R_usb_hhid_TransferEnd	USB data transfer termination request
R_usb_hhid_ChangeDeviceState	Changes the device state.
R_usb_hhid_protcol_code	Gets the protocol code
R_usb_hhid_GetMaxPacketSize	Get the MaxPacketSize
R_usb_hhid_driver_start	HHID driver start processing.
R_usb_hhid_set_report	Sends SET_REPORT
R_usb_hhid_set_protocol	Sends SET_PROTOCOL
R_usb_hhid_transfer_start	USB data transfer request

3.4.1 R_usb_hhid_task

HHID Task

Format

void R_usb_hhid_task(void)

Argument

— —

Return Value

— —

Description

HHID processing task.

HHID task processes requests from the application, and notifies the application of the results.

Note

Call in the scheduler process of the loop.

Example

```
void usb_apl_task_switch(void)
{
  while( 1 )
  {
    /* Scheduler */
    R_usb_cstd_Scheduler();

    if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
    {
       R_usb_hstd_MgrTask();      /* MGR Task */
       R_usb_hhub_Task();         /* HUB Task */
       R_usb_hhid_task();         /* HHID Task */
    }
  }
}
```

3.4.2 R_usb_hhid_class_check

Gets the descriptor information

Format

void R_usb_hhid_class_check(uint16_t **table)

Argument

**table Pointer table

[0] : Device descriptor

[1] : Configuration descriptor

[2] : Interface descriptor

[3] : Result of the descriptor checking

[4] : HUB category

[5] : USB port number

[6] : Transfer speed

[7] : Device address

Return Value

— —

Description

This API is a class driver registration function.

During HHID registration at startup, this function is registered as a callback function in the classcheck member in the driver registration structure, and it is called when a configuration descriptor is received during enumeration.

The function gets the descriptor information for the HID device.

Note

—

Example

```
void  hid_registration(void)
{
  USB_HCDREG_t  driver;

  driver.classcheck   = &R_usb_hhid_class_check;

  R_usb_hstd_DriverRegistration(ptr, (USB_HCDREG_t*)&driver);
}
```

3.4.3 R_usb_hhid_TransferEnd

USB data transfer termination request

Format

USB_ER_t R_usb_hhid_TransferEnd(uint16_t pipe)

Argument

pipe Pipe number

Return Value

USB_OK Success.

USB_ERROR Failure

Description

This function executes a data transfer termination request of specified pipe number to the HCD.

Note

—

Example

```
void  usb_smp_task(void)
{
  USB_ER_t err ;

  /* Transfer end request */
  err = R_usb_hhid_TransferEnd(IN_PIPE);

  return err;
}
```

3.4.4 R_usb_hhid_ChangeDeviceState

Changes device state

Format

void R_usb_hhid_ChangeDeviceState(uint16_t msginfo, uint16_t devadr)

Argument

msginfo USB communication status

devadr Device Address

Return Value

— —

Description

This function changes the device state.

The following values are set to msginfo and change of the USB device State is required of HCD by calling this function.

msginfo	Description
USB_DO_GLOBAL_SUSPEND	Request to change to suspend state
USB_DO_GLOBAL_RESUME	Request to execute resume signal

Note

—

Example

```
void usb_smp_task(void)
{
  /* Change the device state request */
  R_usb_hhid_ChangeDeviceState(USB_DO_GLOBAL_SUSPEND);
}
```

3.4.5 R_usb_hhid_protocol_code

Get the protocol code

Format

uint8_t R_usb_hhid_protocol_code(uint16_t devadr)

Argument

devadr Device address

Return Value

— Protocol code of USB device (bInterfaceProtocol)

Description

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

Note

bInterfaceProtocol is included received Interface Descriptor.

Example

```
void  usb_smp_task(void)
{
  uint8_t   protocol;

  protocol = R_usb_hhid_protocol_code(devadr);
}
```

3.4.6 R_usb_hhid_GetMaxPacketSize

Get the MaxPacketSize

Format

uint16_t R_usb_hhid_GetMaxPacketSize(uint16_t devadr)

Argument

devadr Device address

Return Value

uint16_t MaxPacketSize of Endpoint used when receiving data

Description

Get the MaxPacketSize of the Endpoint to use when receiving data from the device.

Note

MaxPacketSize is included received Endpoint Descriptor.

Example

```
void  usb_smp_task(void)
{
  uint16_t     maxps;

  maxps = R_usb_hhid_GetMaxPacketSize(devadr);
}
```

3.4.7 R_usb_hhid_driver_start

HHID driver start

Format

void	R_usb_hhid_driver_start(void)

Argument

—	—

Return Value

—	—

Description

This function sets the priority of HHID driver task.

The sent and received of message are enable by the priority is set.

Note

Call this function from the user application program during initialization.

Example

```
void usb_hstd_task_start(void)
{
  R_usb_hhid_driver_start();      /* Host Class Driver Task Start Setting */
}
```

3.4.8 R_usb_hhid_set_report

Send SET_REPORT

Format

USB_ER_t R_usb_hhid_set_report(uint16_t devadr,
uint8_t *p_data,
uint16_t length,
USB_UTR_CB_t complete)

Argument

devadr	Device Address
*p_data	Pointer of SET_REPORT data buffer
length	Length of SET_REPORT data
complete	Call-back function

Return Value

USB_OK	Success.
USB_ERROR	Failure

Description

This function sends SET_REPORT to HID device.

Note

—

Example

```
void hid_class_request(uint16_t devadr)
{
  USB_ER_t    err;

  hid_dev_info[devadr].set_report = NUM_LOCK;
  err = R_usb_hhid_set_report( devadr, &hid_dev_info[devadr].set_report,
                        SET_REPORT_LENGTH, &hid_class_request_complete );
}
```

3.4.9 R_usb_hhid_set_protocol

Send SET_PROTOROL

Format

USB_ER_t	R_usb_hhid_set_protocol(uint16_t devadr, uint8_t protocol, USB_UTR_CB_t complete)

Argument

devadr	Device Address
protocol	SET_PROTOCOL data(0:Boot, 1:Report)
complete	Call-back function

Return Value

USB_OK	Success.
USB_ERROR	Failure

Description

This function sends SET_PROTOCOL to HID device.

Note

—

Example

```
void hid_class_request(uint16_t devadr)
{
  USB_ER_t    err;

  err = R_usb_hhid_set_protocol(devadr, BOOT_PROTOCOL, &hid_class_request_complete);
}
```

3.4.10 R_usb_hhid_transfer_start

USB data transfer request

Format

USB_ER_t	R_usb_hhid_transfer_start(uint8_t *table, uint32_t size, USB_CB_t complete, uint16_t pipe)

Argument

*table	Pointer to the data buffer area
size	Read data size
complete	Call-back function
pipe	Pipe number

Return Value

USB_OK	Success
USB_ERROR	Failure

Description

This function requests a data transfer to the USB device with specified pipe.

When data transfer ends (specified data size reached, short packet received, error occurred), the call-back function is called.

The argument of this callback function is stored transfer results.

Note

—

Example

```
uint8_t  buf[8];

USB_ER_t usb_smp_task(void)
{
  USB_ER_t err;

  err = R_usb_hhid_transfer_start(&buf, 8, usb_data_received, IN_PIPE);

  return err;
}

/* Callback function */
void usb_data_received(USB_UTR_t *utr)
{
}
```

4. Sample Application

This section describes the initial settings necessary for using the HHID and USB-BASIC-F/W in combination as a USB driver and presents an example of data transfer by means of processing by the main routine and the use of API functions.

4.1 Operating Environment

Figure 4-1 shows an example operating environment for the HHID.

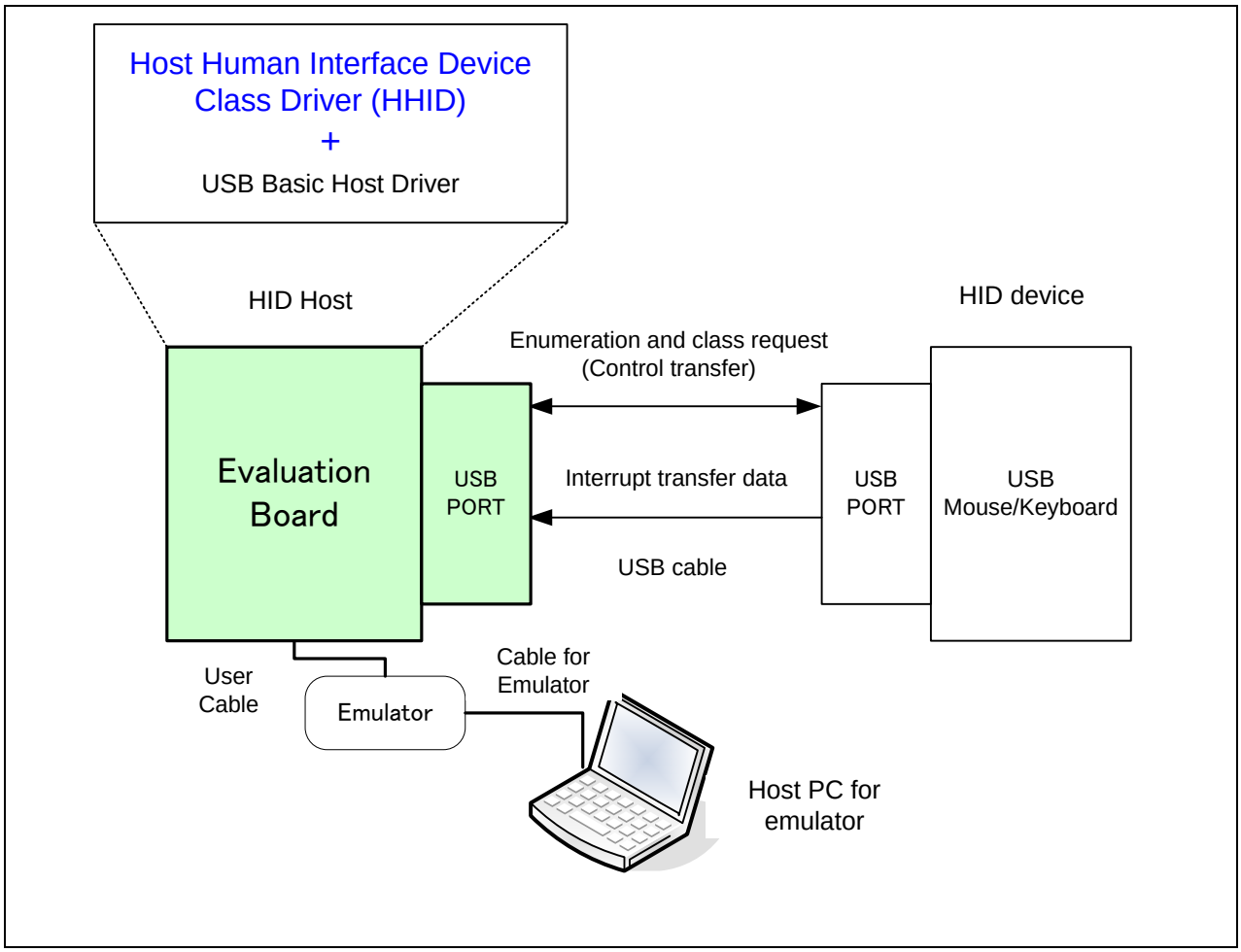


Figure 4-1 Example Operating Environment

4.2 Application Specifications

The main functions of the HHID sample application are as follows:

1) Transferring data to and from the HID device (mouse or keyboard)

The APL transfers data to and from the connected HID device. By using a USB hub it is possible to connect up to some HID devices, and the APL can transfer data to and from some devices at the same time.

a) When a mouse is connected, turns on the LEDs when the mouse buttons are pressed.

Left click	: LED0
Right click	: LED1
Center click	: LED2
Move	: LED3

b) When a keyboard is connected, turn on the LEDs when the input key data at a time.

Alphabet	: LED1
Number	: LED2
Other	: LED0

2) Suspending and resuming the HID device (When “USB_HHID_SUSPEND_ENABLE” is defined.)

When certain period of time has elapsed with no operation state, it will suspend to the connected HID device.
When certain period of time has elapsed in the suspend state, it will resume to the connected HID device.

Note: Suspend and resume only take place when the HID device is connected directly to the USB port.
Suspend and resume are not supported for HID devices connected via a hub.
Time is measured by software. Interval is set in the definition value of IDLE_LOOP_COUNT.

4.3 Initial Settings of USB Driver

Sample settings are shown below.

```
void usb_hhid_apl(void)
{
   /* MCU Pin Setting (Refer to “4.3.1”) */
   usb_mcu_setting();

   /* USB Driver Setting (Refer to “4.3.2”) */
   R_usb_hstd_MgrOpen();
   R_usb_cstd_SetTaskPri(USB_HUB_TSK, USB_PRI_3);    // Note
   R_usb_hhub_Registration(USB_NULL);                // Note
   hid_registration();
   R_usb_hhid_driver_start();

   /* Main routine */
   usb_hapl_mainloop();
}
```

[Note]
It is only necessary to call this function when the HUB will be used.

4.3.1 MCU Settings

Set the USB module according to the initial setting sequence of the hardware manual, the USB interrupt handler registration and USB interrupt enable setting.

4.3.2 USB Driver Settings

The USB driver settings consist of registering a task with the scheduler and registering class driver information for the USB-BASIC-F/W. The procedure is described below.

1. Call the USB-BASIC-F/W's API function (R_usb_hstd_MgrOpen() to register the MGR task and the HCD task with the scheduler.
2. Call the class driver API function (R_usb_hhub_Registration()) to register the HUB task with the scheduler.
3. After specifying the necessary information in the members of the class driver registration structure (USB_HCDREG_t), call the USB-BASIC-F/W's API function (R_usb_hstd_DriverRegistration() to register the class driver information.
4. Call the class driver HHID's API function (R_usb_hhid_driver_start()) to register the HHID task with the scheduler.

A sample of information specified in the structure declared by USB_HCDREG_t is shown below.

```
void usb_hapl_registration(void)
{
   /* Structure for the class driver registration */
   USB_HCDREG_t  driver;

   /* Class Code which is defined in the USB specification setting*/
   driver.ifclass     = (uint16_t)USB_IFCLS_HID;
   /* TPL setting */
   driver.tpl         = (uint16_t*)&usb_gapl_devicetpl; // Note 1
   /* Set the class check function which is called in the enumeration. */
   driver.classcheck  = &R_usb_hhid_class_check;
   /* Set the function which is called when completing the enumeration */
   driver.devconfig   = &hid_configured;
   /* Set the function which is called when disconnecting USB device */
   driver.devdetach   = &hid_detach;
   /* Set the function which is called when changing the suspend state */
   driver.devsuspend  = &hid_suspend;
   /* Set the function which is called when resuming from the suspend state */
   driver.devresume   = &hid_resume;

   /* Register the class driver information to HCD */
   R_usb_hstd_DriverRegistration(&driver);
}
```

[Note]

1. TPL(Target Peripheral List) need to be defined in the application program. Refer to USB Basic Firmware application note (Document No.R01AN2633EJ) about TPL.

4.4 Processing by Main Routine

After the USB driver initial settings, call the scheduler (R_usb_cstd_Scheduler()) from the main routine of the application. Calling R_usb_cstd_Scheduler() from the main routine causes a check for events. If there is an event, a flag is set to inform the scheduler that an event has occurred. After calling R_usb_cstd_Scheduler(), call R_usb_cstd_CheckSchedule() to check for events. Also, it is necessary to run processing at regular intervals to get events and perform the appropriate processing.*[1]

```
void usb_hapl_mainloop(void)
{
  while(1)  // Main routine
  {
    // Confirming the event and getting (Note 1)
    R_usb_cstd_Scheduler();

    // Judgment whether the event is or not
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_MgrTask();   // MGR task
        R_usb_hhub_Task();      // HUB task (Note 3)
        R_usb_hhid_task();      // HID task
    }
    hhid_application();  // User application program
  }
}
```

(Note2)

[Note]

1. If, after getting an event with R_usb_cstd_Scheduler() and before running the corresponding processing, R_usb_cstd_Scheduler() is called again and gets another event, the first event is discarded. After getting an event, always call the corresponding task to perform the appropriate processing.
2. Be sure to describe these processes in the main loop for the application program.
3. It is only necessary to call this function when the HUB will be used.

4.4.1 Application Processing

The application comprises two parts: initial settings and main loop. An overview of the processing in these two parts is provided below.

1. The APL manages the states and the events associated with them. The APL first checks the state of the connected device (see Table 4-1). It then stores the state as a member of a structure (see 4.4.2) managed by the APL.
2. The APL confirms the event (see Table 4-2) associated with the state and performs the processing corresponding to the event. After event processing, the APL changes the state if necessary. Note that the event is stored as the member of a structure (see 4.4.2) managed by the APL.

An overview of the processing performed by the APL is shown below:

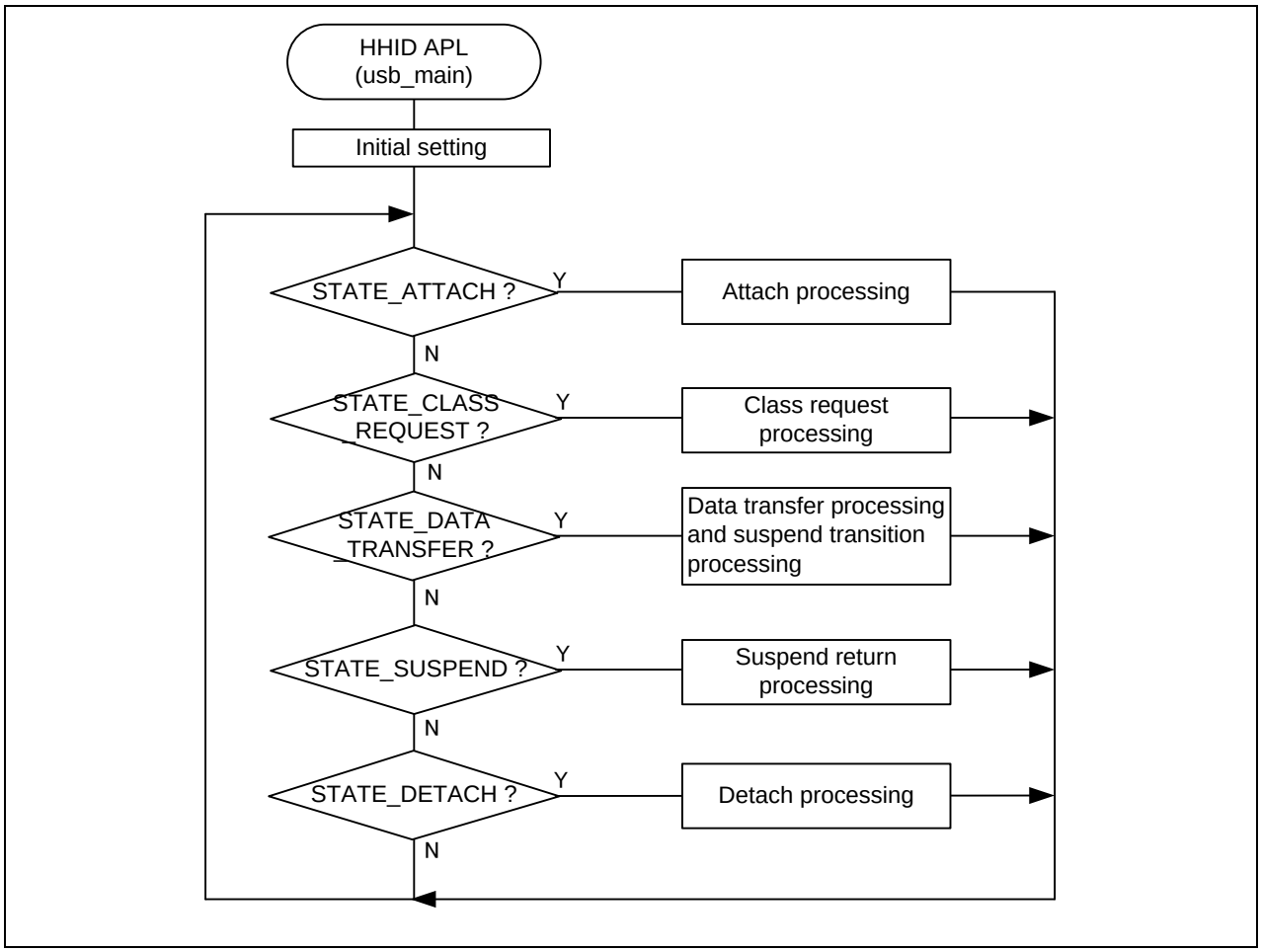


Figure 4-2 Main Loop

4.4.2 State and Event Management

Members of the following structure are used to manage states and events. This structure is prepared by the APL, and it is also used to manage the device addresses of connected HID devices.

If the processing to get the event determines that the event to be fetched is not present, the event is set to "EVENT_NONE."

```
/* Structure for HID device control */
typedef struct DEV_INFO
{
    uint16_t    state;                /* State for application */
    uint16_t    event_cnt;            /* Event count */
    uint16_t    event[EVENT_MAX];     /* Event no. */
    uint16_t    pipe;                 /* Use pipe no. */
    uint16_t    protocol;             /* HID Protocol(Mouse/Keybord/none) */
    uint8_t     cr_seq;               /* Class Request Sequence */
    uint8_t     set_report;           /* SetReprot (Send)DATA */
    uint8_t     data[REPORT_LENGTH];  /* Receive Data */
}
DEV_INFO_t;
```

Table 4-1 List of States

State	State Processing Overview	Related Event
STATE_ATTACH	Attach processing	EVENT_CONFIGURD
STATE_CLASS_REQUEST	Class request processing	EVENT_CLASS_REQUEST_START
		EVENT_USB_TRANSFER_COMPLETE
STATE_DATA_TRANSFER	Data transfer processing and suspend transition processing	EVENT_USB_TRANSFER_START
		EVENT_USB_TRANSFER_COMPLETE
		EVENT_SWITCH_INPUT
STATE_SUSPENDED	Suspend return processing	EVENT_SWITCH_INPUT
STATE_DETACH	Detach processing	--

Table 4-2 List of Events

Event	Outline
EVENT_CONFIGURD	USB device connecting completion
EVENT_CLASS_REQUEST_START	Request of send the class request
EVENT_CLASS_REQUEST_COMPLETE	Class request complete
EVENT_USB_TRANSFER_START	Data transfer request
EVENT_USB_TRANSFER_COMPLETE	Data transfer complete
EVENT_SWITCH_INPUT	Switch input
EVENT_NONE	No event

An overview of the processing associated with each state is provided below.

1) Attach processing (STATE_ATTACH)

== Outline ==

In this state, processing is performed to notify that a HID device has connected and that enumeration has finished, and the state changes to STATE_CLASS_REQUEST.

== Description ==

(1) In the APL, first the initialization function sets the state to STATE_ATTACH and the event to EVENT_NONE.

(2) The state continues to be STATE_ATTACH until an HID device is connected, and *hid_connect_wait()* is called. When an HID device is connected and enumeration completes, the callback function *hid_configured()* is called by the USB driver, This callback function issues the event EVENT_CONFIGURD. This callback function *hid_configured()* is specified in the member *devconfig* of structure *USB_HCDREG_t*.

(3) In EVENT_CONFIGURD, the *hid_connect_wait()* performs the following processing, and class request processing starts:

a) Sets the information of the connected HID device in the variable (*hid_dev_info*).

b) Changes the state to STATE_CLASS_REQUEST and the event EVENT_CLASS_REQUEST_START is issued.

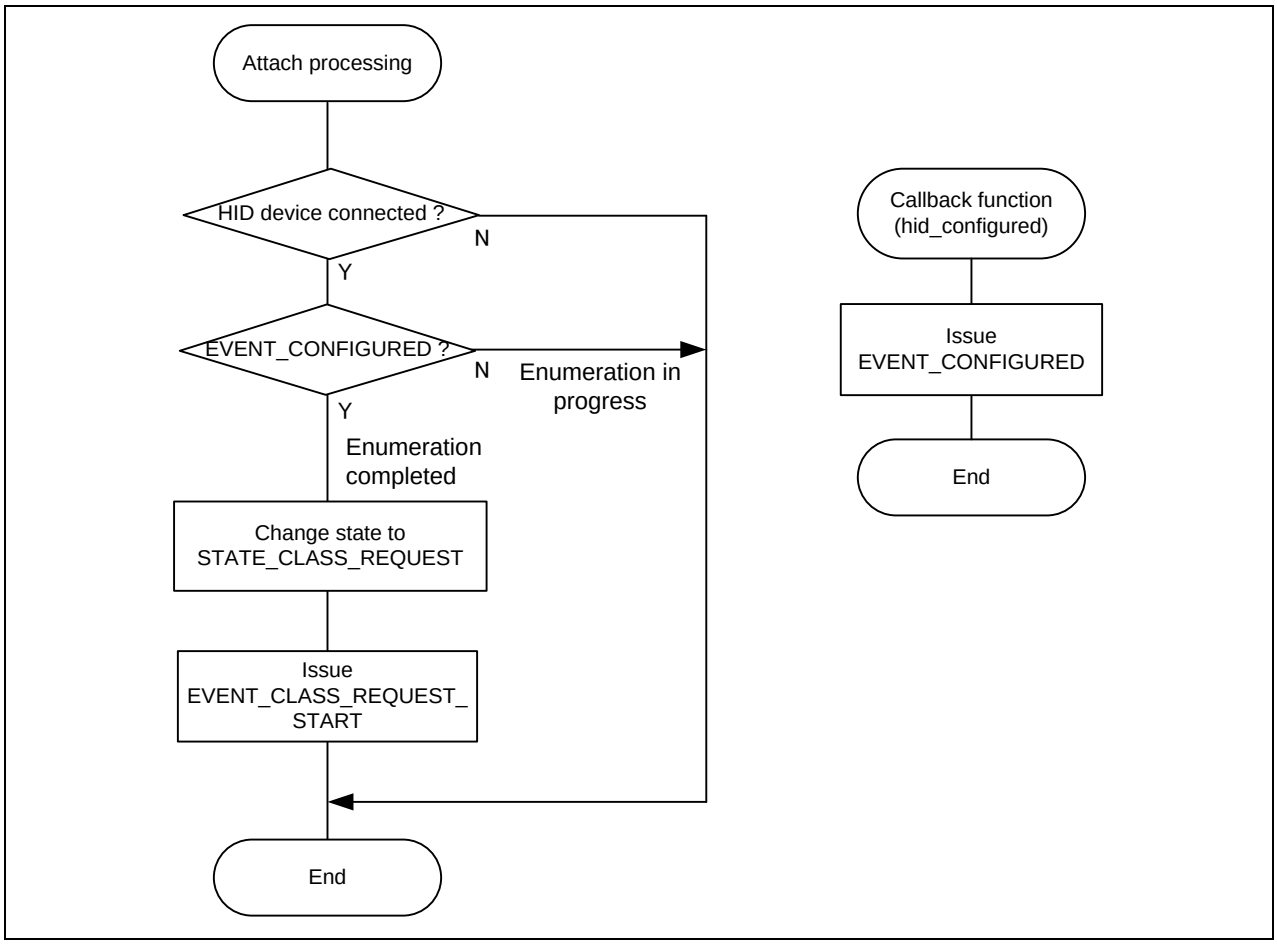


Figure 4-3 Flowchart of Attach Processing

2) Class Request Processing (STATE_CLASS_REQUEST)

== Outline ==

In this state, processing is performed to transmit class requests to the HID device. When transmission of a separately specified class requests finishes, the state changes to STATE_DATA_TRANSFER.

== Description ==

(1) In this state, first EVENT_CLASS_REQUEST_START is processed by the *hid_class_request()* function, and a class request transmit request is sent to the USB driver.

(2) When class request transmit processing completes, the callback function *hid_class_request_complete()* is called. This callback function issues the event EVENT_CLASS_REQUEST_COMPLETE.

(3) In EVENT_CLASS_REQUEST_COMPLETE, the processing branches to one of the following according to the class request transmitted in (1). above:

a) If the class request *SetProtocol* was transmitted to a mouse, the state is changed to STATE_DATA_TRANSFER and the event to EVENT_USB_TRANSFER_START. Data transfer processing starts in the next processing loop.

b) If the class request *SetProtocol* was transmitted to a keyboard, the event EVENT_CLASS_REQUEST_START is issued, and class request *SetReport* transmit processing starts.

c) If the class request *SerReport* was transmitted, the state is changed to STATE_DATA_TRANSFER and the event EVENT_USB_TRANSFER_START is issued.

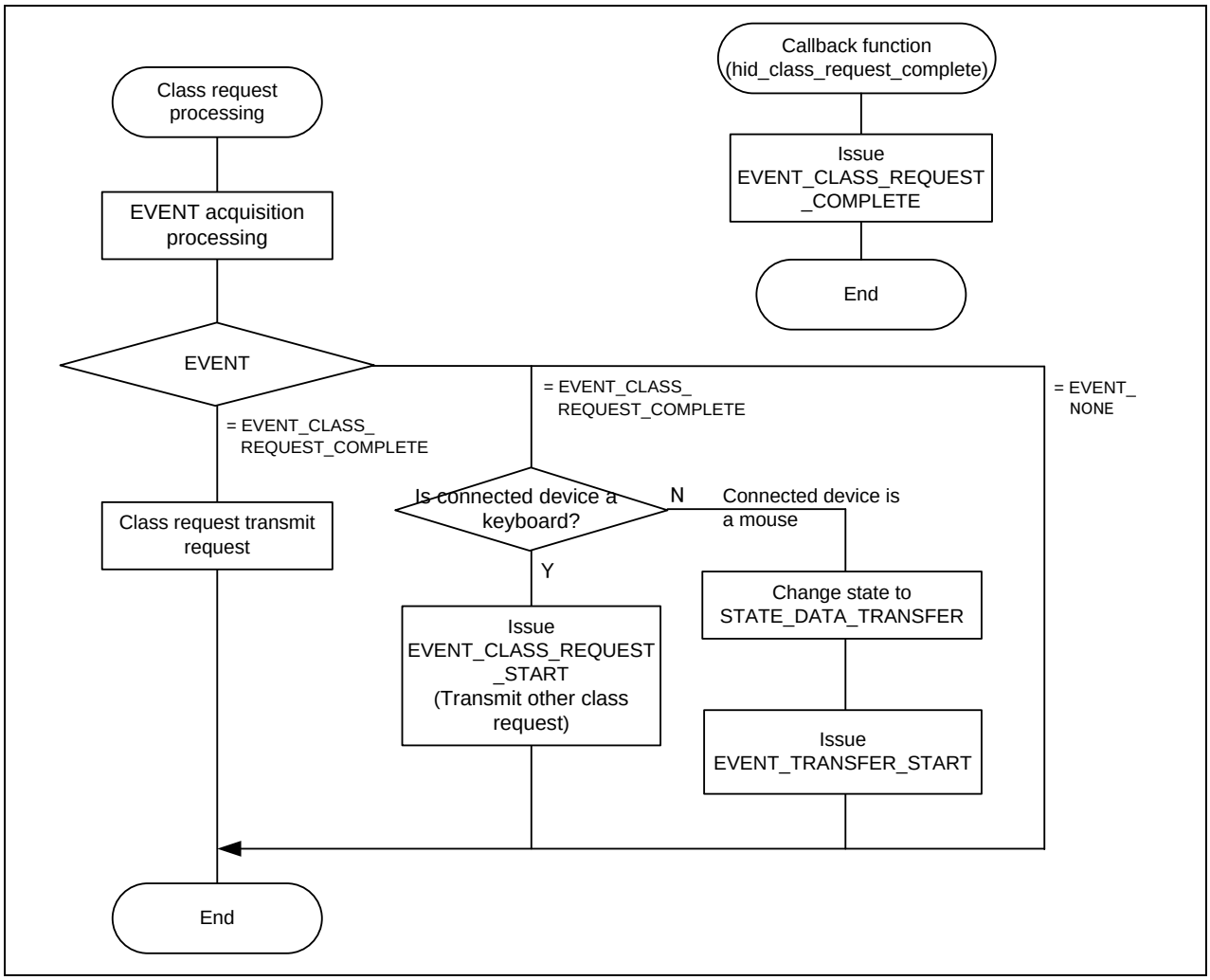


Figure 4-4 Flowchart of Class Request Processing

3) Data Transfer Processing and Suspend Transition Processing (STATE_DATA_TRANSFER)

== Outline ==

In this state, processing is performed to receive data from the HID device and to process data. In addition, processing is performed to transmit a SUSPEND signal to the HID device.

== Description ==

(1) In this state, EVENT_USB_TRANSFER_START is processed by the *hid_data_transfer()*, and a data transfer processing request is sent to the USB driver.

(2) When data transfer processing completes, the callback function *hid_receive_complete()* is called. This callback function issues the event EVENT_USB_TRANSFER_COMPLETE.

(3) In EVENT_USB_TRANSFER_COMPLETE, the *hid_data_transfer()* function is in the process corresponding received data, then the event is set to EVENT_USB_TRANSFER_START.

(4) If suspend during steps (1) to (3), above, of data transfer, the state is changed to STATE_SUSPEND, and the *R_usb_hhid_ChangeDeviceState()* function is used to transmit a SUSPEND signal to the connected HID device.

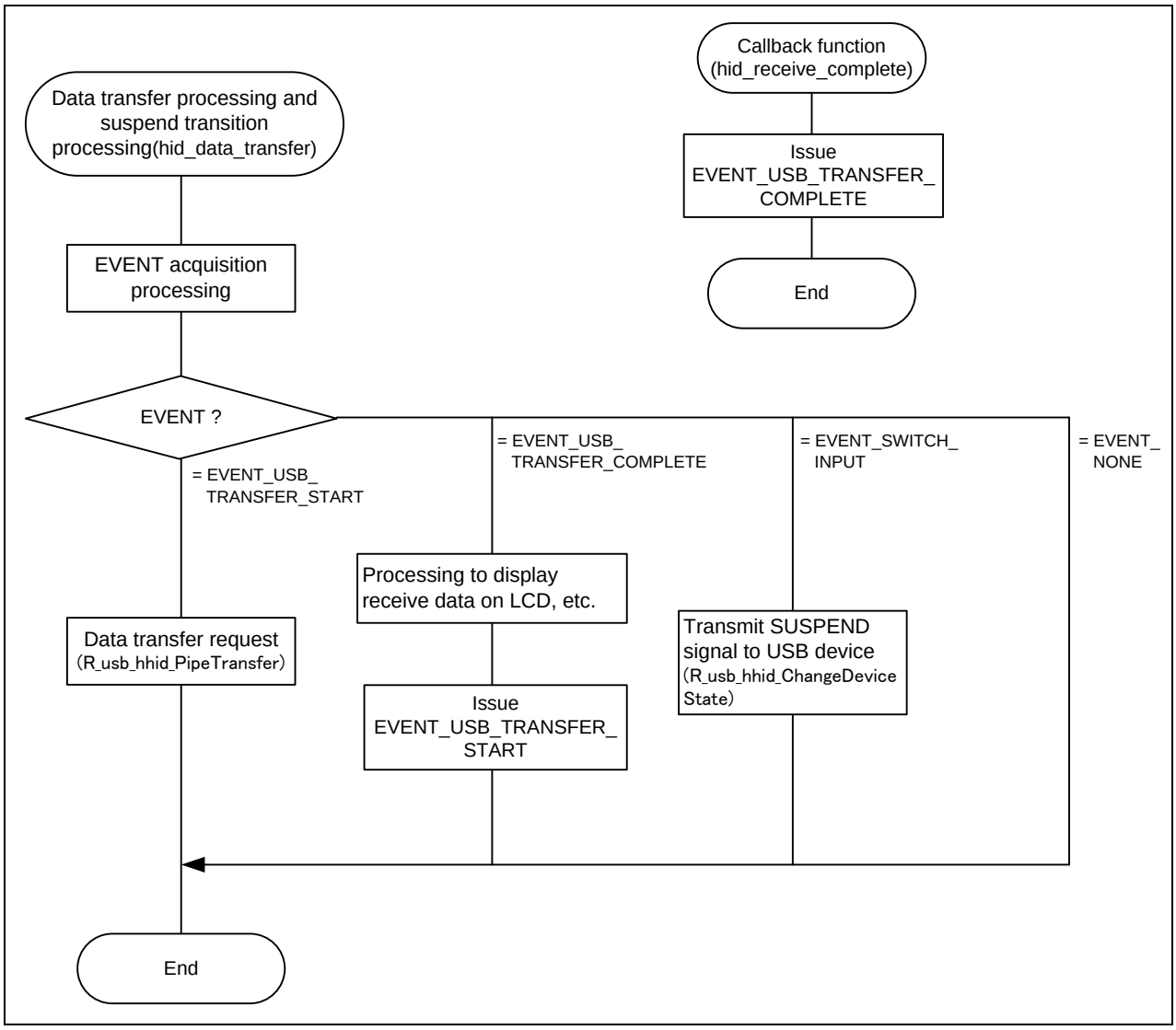


Figure 4-5 Flowchart of Data Transfer Processing and Suspend Transition Processing

4) Suspend Return Processing (STATE_SUSPENDED)

== Outline ==

In this state, processing is performed to transmit a RESUME signal to the HID device when in suspend status to wake it from suspend status. When transmission of the RESUME signal finishes, the callback function *hid_resume()* is called to notify.

== Description ==

(1) After a period time, the *hid_suspended()* function uses the *R_usb_hhid_ChangeDeviceState()* function to transmit aRESUME signal to the connected HID device.

(2) When transmission of the RESUME signal completes, the callback function *hid_resume()* is called by the USB driver. This callback function changes the state to STATE_DATA_TRANSFER and issues the event EVENT_USB_TRANSFER_START.

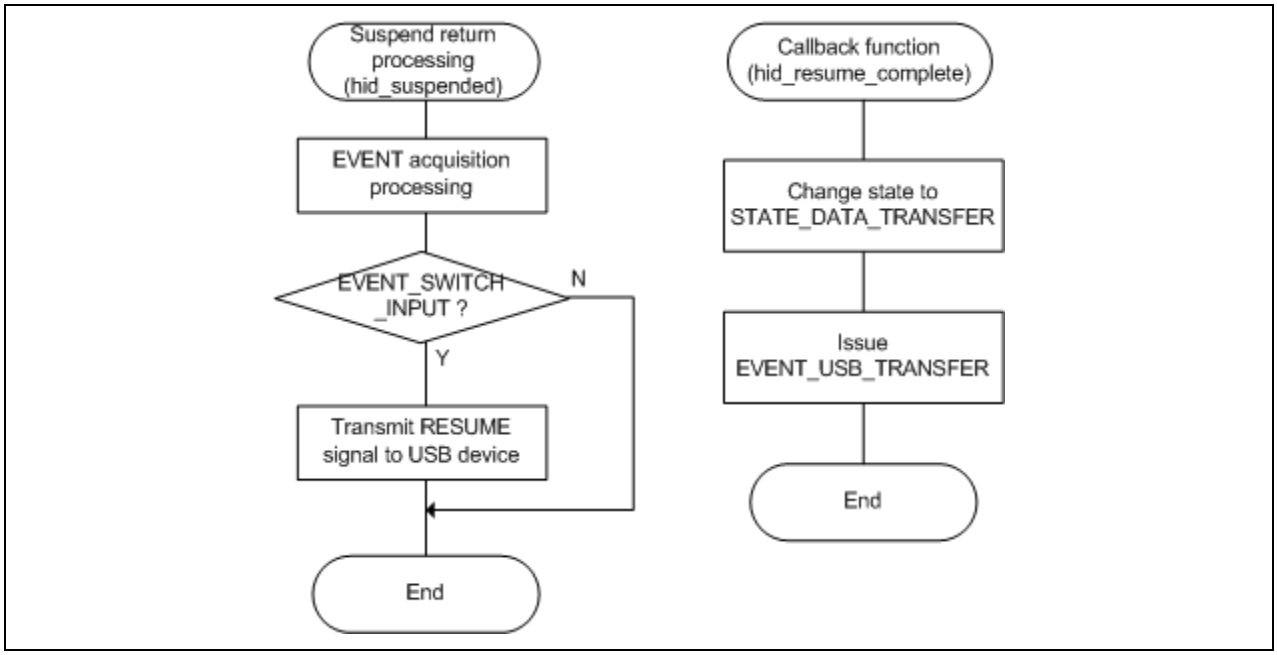


Figure 4-6 Flowchart of Suspend Return Processing

5) Detach Processing (STATE_DETACH)

When the connected HID device is disconnected, the USB driver calls the callback function *hid_detach()*. This callback function changes the state to STATE_DETACH. In STATE_DETACH, processing is performed to clear the variables and change the state to STATE_ATTACH, among other things.

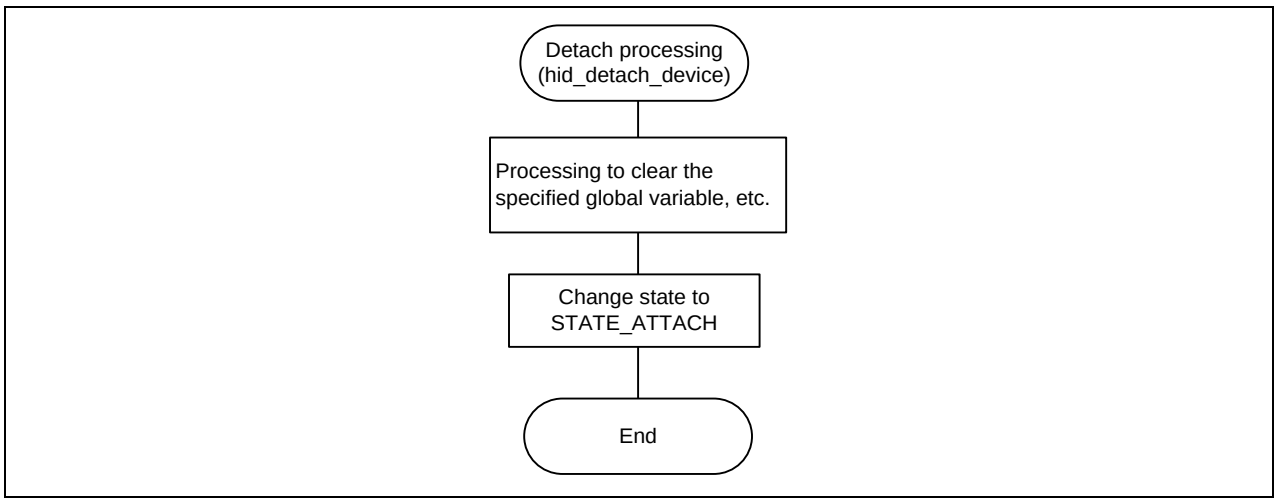


Figure 4-7 Flowchart of Detach Processing

Appendix A. Changes of initial setting

USB-BASIC-F/W has been changed to "RZ/T1 group initial setting Rev.1.30".
Sample program supports IAR embedded workbench for ARM (EWARM), DS-5 and e^2 studio.
This chapter describes the changes.

Folders and files

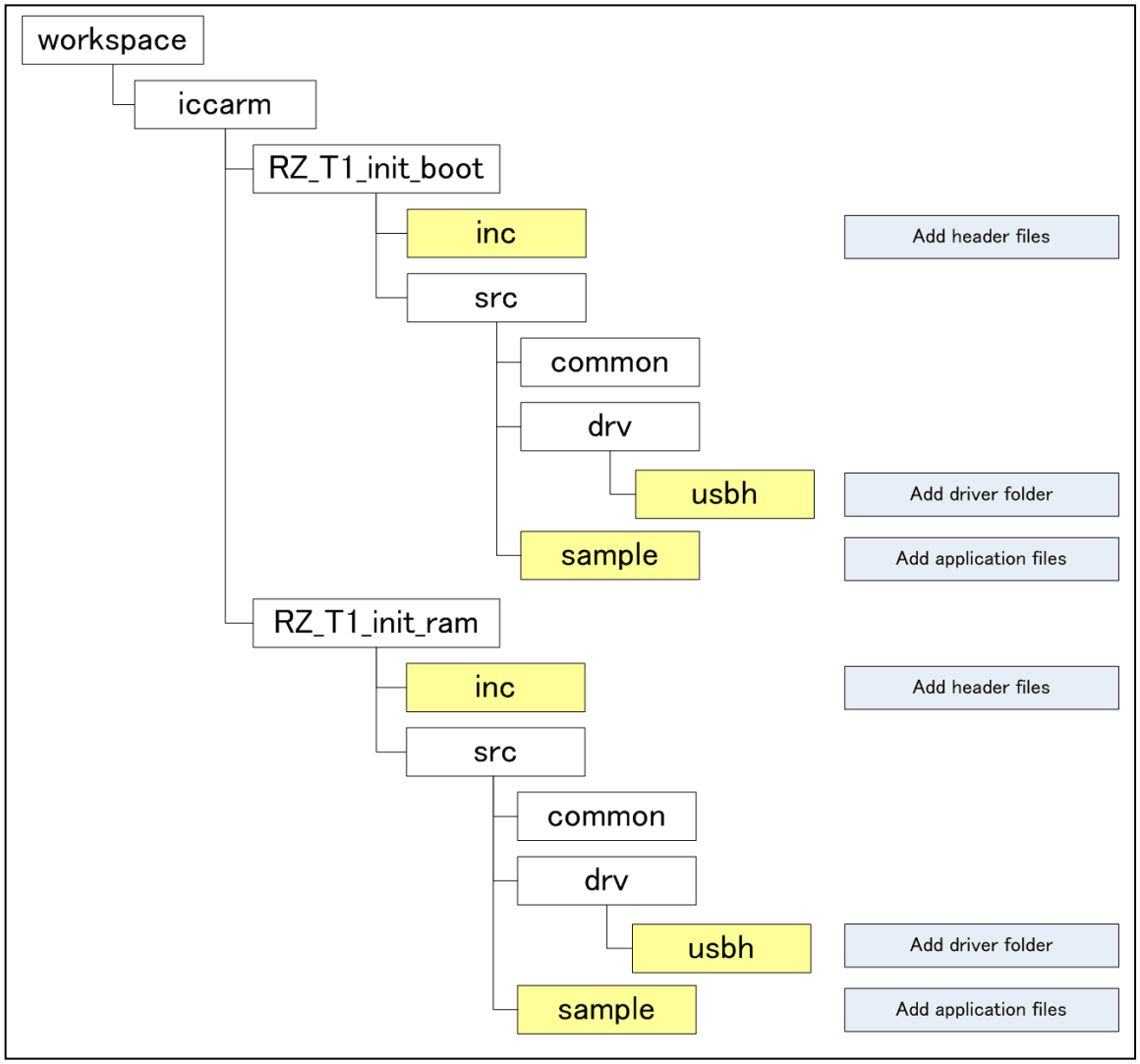
In the "RZ/T1 group initial setting Rev.1.30", different folder structure by the development environment and the boot method. Changes to each folder of all of the development environment and the boot method it is shown below.

- Add the following files in the “inc” folder.
  r_usb_basic_config.h
  r_usb_basic_if.h
  r_usb_hhid_config.h
  r_usb_hhid_if.h

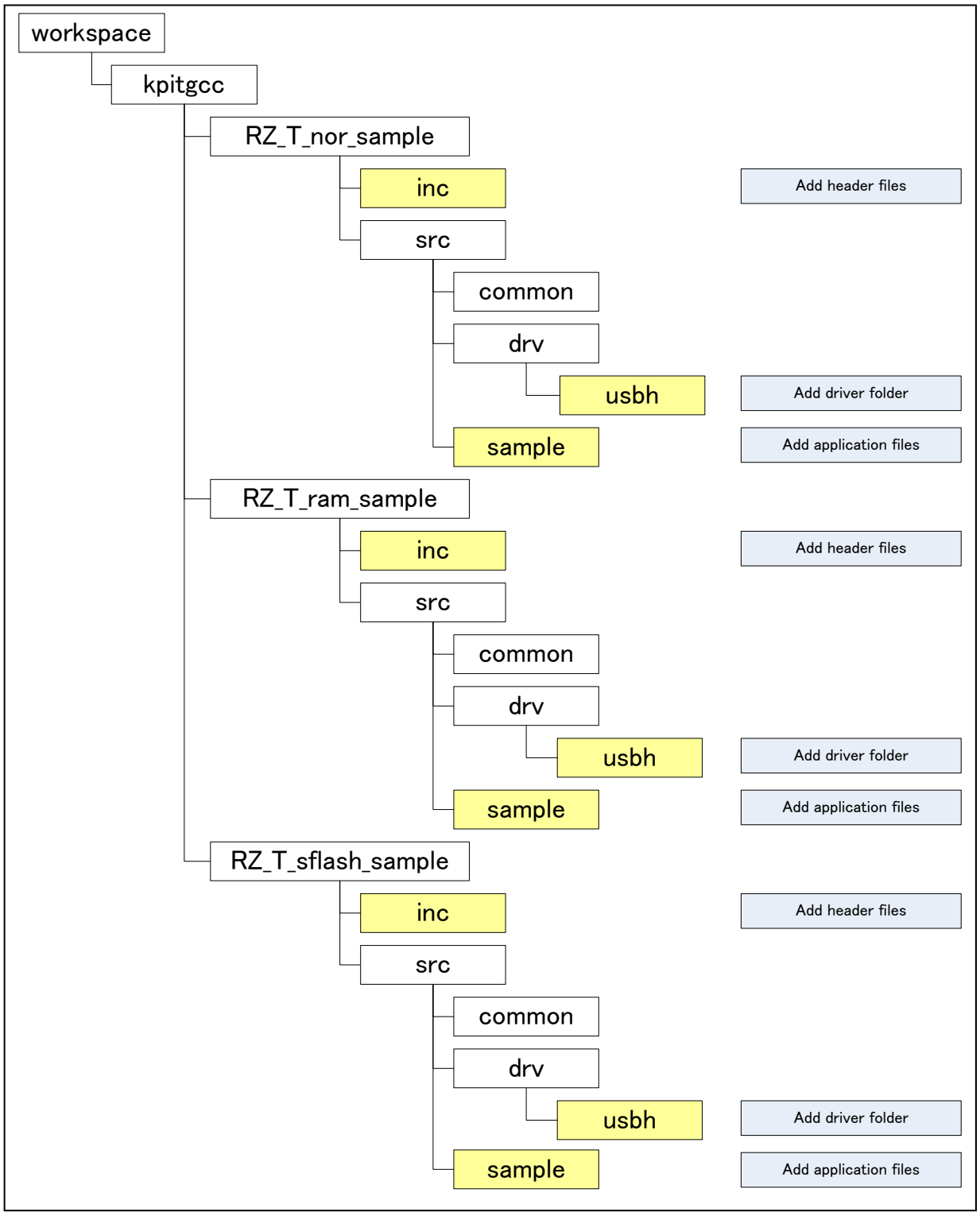
- Add the following files in the “sample” folder.
  r_usb_main.c
  r_usb_hhid_apl.c
  r_usb_hhid_apl.h

- Add the “usbh” folder and the following files “usbh” folder in the “drv” folder.

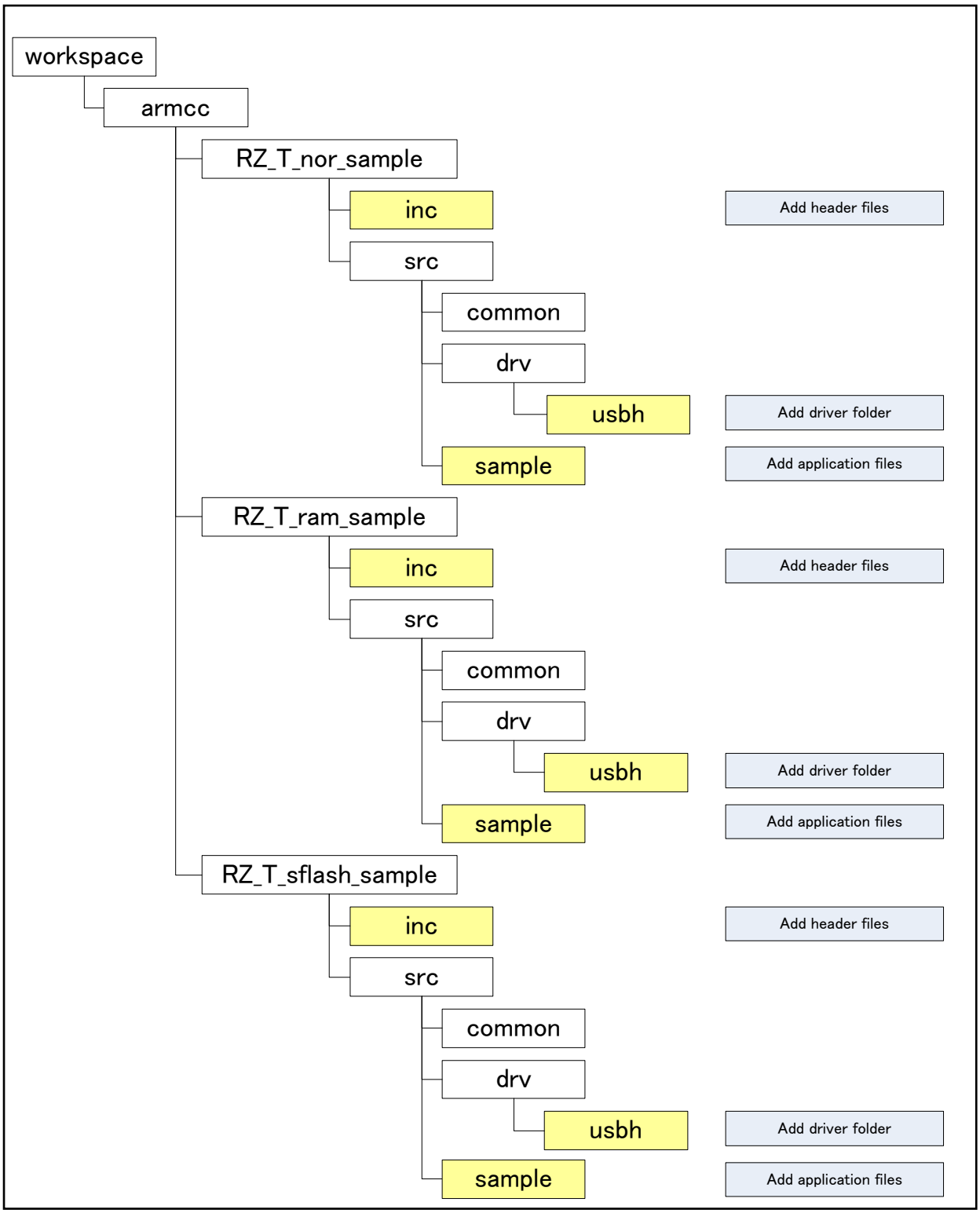
The following is the folder structure of EWARM.



The following is the folder structure of e^2 studio.



The following is the folder structure of DS-5.



Section

Modify the section size of the code area and a data area, and add the following section.

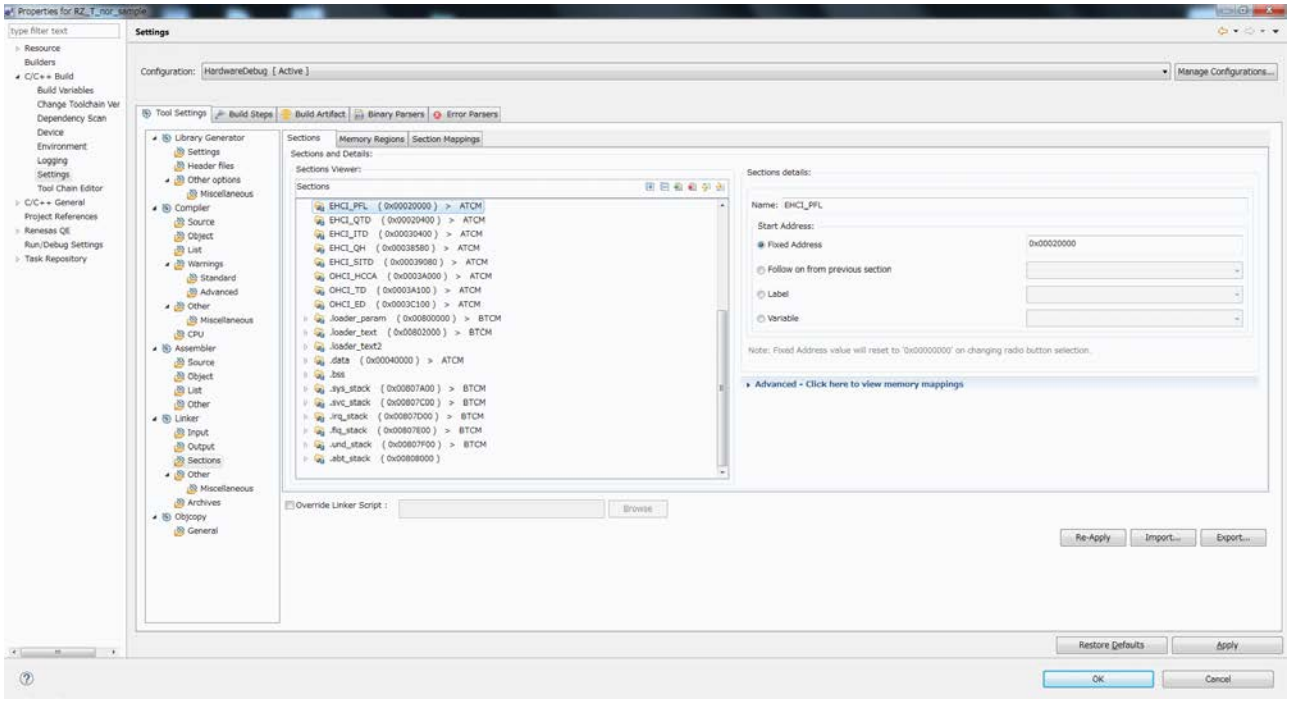
Section name	Address	variable	file
EHCI_PFL	0x00020000	ehci_PeriodicFrameList	r_usb_hEhciMemory.c
EHCI_QTD	0x00020400	ehci_Qtd	
EHCI_ITD	0x00030400	ehci_Itd	
EHCI_QH	0x00038580	ehci_Qh	
EHCI_SITD	0x00039080	ehci_Sitd	
OHCI_HCCA	0x0003A000	ohci_hcca	r_usb_hOhciMemory.c
OHCI_TD	0x0003A100	ohci_TdMemory	
OHCI_ED	0x0003c100	ohci_EdMemory	

e² studio

e² studio sets the section in the configuration screen.
Changes are as follows:

- Fixed address of ".data" section from 0x0007F000 to 0x00040000
- Add section setting of EHCI and OHCI.

Refer to [Project] → [Properties] → [C/C++ Build] → [Settings] → [Sections].



Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __GNUC__
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ]
                __attribute__ ((section ("EHCI_PFL")));
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ]
                __attribute__ ((section ("EHCI_QH")));
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ]
                __attribute__ ((section ("EHCI_QTD")));
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ]
                __attribute__ ((section ("EHCI_ITD")));
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ]
                __attribute__ ((section ("EHCI_SITD")));
#endif  /* __GNUC__ */
```

r_usb_hOhciMemory.c

```
#ifdef __GNUC__
static USB_OHCI_HCCA_BLOCK              ohci_hcca
                __attribute__ ((section ("OHCI_HCCA")));
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR ohci_TdMemory[USB_OHCI_NUM_TD]
                __attribute__ ((section ("OHCI_TD")));
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR ohci_EdMemory[USB_OHCI_NUM_ED]
                __attribute__ ((section ("OHCI_ED")));
#endif  /* __GNUC__ */
```

EWARM

EWARM sets the section in the linker setting file (.icf file).

Changes are as follows:

・Start address of RAM region from 0x00070000 to 0x00040000.

・End address of USER_PRG region from 0x0006FFFF to 0x0001FFFF.

```
define symbol __ICFEDIT_region_RAM_start__   = 0x00040000;

define symbol __region_USER_PRG_end__        = 0x0001FFFF;
```

・To the EHCI and OHCI to fixed address, adds memory region definition.

```
define region EHCI_MEM1_region = mem:[from 0x00020000 to 0x000203FF];
define region EHCI_MEM2_region = mem:[from 0x00020400 to 0x00039FFF];

define region OHCI_MEM1_region = mem:[from 0x0003A000 to 0x0003A0FF];
define region OHCI_MEM2_region = mem:[from 0x0003A100 to 0x0003FFFF];

do not initialize  { section EHCI_PFL, section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section
EHCI_SITD, section OHCI_HCCA, section OHCI_TD, section OHCI_ED };

place in EHCI_MEM1_region { section EHCI_PFL };
place in EHCI_MEM2_region { section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section EHCI_SITD };

place in OHCI_MEM1_region { section OHCI_HCCA };
place in OHCI_MEM2_region { section OHCI_TD, section OHCI_ED };
```

Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __ICCARM__
#pragma location="EHCI_PFL"
static uint32_t                 ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma location="EHCI_QH"
static USB_EHCI_QH              ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma location="EHCI_QTD"
static USB_EHCI_QTD             ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma location="EHCI_ITD"
static USB_EHCI_ITD             ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma location="EHCI_SITD"
static USB_EHCI_SITD            ehci_Sitd[ USB_EHCI_NUM_SITD ];
#endif  /* __ICCARM__ */
```

r_usb_hOhciMemory.c

```
#ifdef __ICCARM__
#pragma location="OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma location="OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR     ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma location="OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR     ohci_EdMemory[USB_OHCI_NUM_ED];
#endif  /* __ICCARM__ */
```

DS-5

DS-5 sets the section in the linker setting file (scatter file).

Changes are as follows:

・Start address of RAM region from 0x00040000and BSS region(0clear init memory region) follow RAM region.

```
DATA            0x00040000    UNINIT
{
    * (+RW)
}
BSS             +0
{
    * (+ZI)
}
```

・To the EHCI and OHCI to fixed address, adds memory region definition.

```
    EHCI_PERIODIC_FRAMELIST 0x00020000    0x400
    {
            r_usb_hEhciMemory.o(EHCI_PFL)
    }
    EHCI_QTD          +0
    {
            r_usb_hEhciMemory.o(ehci_Qtd)
    }
    EHCI_ITD          +0
    {
            r_usb_hEhciMemory.o(ehci_Itd)
    }
    EHCI_QH           +0
    {
            r_usb_hEhciMemory.o(ehci_Qh)
    }
    EHCI_SITd         +0
    {
            r_usb_hEhciMemory.o(ehci_Sitd)
    }
    OHCI_HCCA         0x0003A000      0x100
    {
            r_usb_hOhciMemory.o(OHCI_HCCA)
    }
    OHCI_TDMEMORY     +0
    {
            r_usb_hOhciMemory.o(OHCI_TD)
    }
    OHCI_EDMEMORY     +0
    {
            r_usb_hOhciMemory.o(OHCI_ED)
    }
```

Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "EHCI_PFL"
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QH"
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QTD"
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_ITD"
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_SITD"
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ];
#pragma arm section zidata
#endif
```

r_usb_hOhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma arm section zidata
#pragma arm section zidata = "OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR     ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma arm section zidata
#pragma arm section zidata = "OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR     ohci_EdMemory[USB_OHCI_NUM_ED];
#pragma arm section zidata
#endif
```

Call the USB-BASIC-FW function

Adds the usbh_main() of USB-BASIC-F/W in the main() of “\src\sample\int_main.c”.

```
extern void usbh_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBh main */
    usbh_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

Website and Support

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

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Aug 21, 2015	—	First edition issued
1.10	Dec 25, 2015	29	Added Appendix A
1.20	Feb 29, 2016	31,35,36	Added DS-5 setting
1.30	Dec 07, 2017	—	Corresponds to RZ / T1 initial setting Ver 1.30
		2	Added about HID device which performs Interrupt OUT transfer as Limitations
		6,12	Added description of R_usb_hhid_GetMaxPacketSize

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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 Microprocessing unit or Microcontroller unit products 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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

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 6.0