R01AN3466EJ0100
Rev.1.00
Sep 30, 2016

RZ/A1H Group

USB Host Human Interface Device Class Driver (HHID)

Introduction

This application note describes USB Host Human Interface Device Class Driver (HHID). This driver operates in combination with the USB Basic Host Driver (USB-BASIC-F/W). It is referred to below as the HHID.

Target Device

RZ/A1H 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
   http://www.usb.org/developers/docs/
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/A1H Group,RZ/A1H Group User's Manual: Hardware (Document No.R01UH0403EJ)
5. RZ/A1H Group USB Host and Peripheral Interface Driver (Document No.R01AN3291EJ)
6. RZ/A1H Group Downloading Program to NOR Flash Memory Using ARM® Development Studio 5 (DS-5™) Semihosting Function (for GENMAI) (Document No.R01AN1957EJ)
7. RZ/A1H Group I/O definition header file (Document No.R01AN1860EJ)
8. RZ/A1H Group Example of Initialization (for GENMAI) (Document No.R01AN1864EJ)

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

Contents

1. Overview

The HHID, when used in combination with the USB-BASIC-F/W, operates as a USB host human interface device class driver (HHID).
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
- HCDC can connect maximum 3 HID devices to 1 USB channel by using USB Hub.

1.1 Please be sure to read

It is recommended to use the APIs described in the document (Document No: R01AN3291EJ) when creating an application program using this driver.
That document is located in the "reference_documents" folder within the package.
[Note]

a. The document (Document No: R01AN3291EJ) also provides how to create an application program using the APIs described above.
b. If the APIs described in the document (Document No: R01AN3291EJ) are used, there is no need to use the API described in “7.2. HHID API Functions” of this document of this document.

1.2 Operation Confirmation Conditions

The operation of the USB Driver module has been confirmed under the conditions listed in Table 1.1.

Table 1.1 Operation Confirmation Conditions

Item	Description
MCU	RZ/A1H
Operating frequency (Note)	CPU clock (Iφ): 400 MHz
	Image-processing clock (Gφ): 266.37 MHz
	Internal bus clock (Bφ): 133.33 MHz
	Peripheral clock 1 (P1φ): 66.67 MHz
	Peripheral clock 0 (P0φ): 33.33 MHz
Operating voltage	Power supply voltage (I/O): 3.3 V
	Power supply voltage (internal): 1.8 V
Integrated development environment	ARM Integrated Development Environment
	• ARM Development Studio (DS-5™) Version 5.16
	IAR Integrated Development Environment
	• IAR Embedded Workbench for ARM Version 7.40
Compiler	ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]
	KPIT GNUARM-RZ v14.01
	IAR C/C++ Compiler for ARM 7.40
Operating mode	Boot mode 0
	(CS0-space 16-bit booting)
Communication setting of terminal software	Communication speed: 115200 bps
	Data length: 8 bits
	Parity: None
	Stop bit length: 1 bit
	Flow control: None
Board	GENMAI board
	R7S72100 CPU board (RTK772100BC00000BR)
Device (Functions used on the board)	Serial interface (D-sub 9-pin connector)
	USB1 connector, USB2 connector

1.3 Limitations

The following limitations apply to the HHID.

1. 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.)

Terms and Abbreviations

APL	:	Application program
cstd	:	Prefix of function and file for Peripheral & Host Common Basic (USB low level) F/W
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
hstd	:	Prefix of function and file for Host Basic (USB low level) F/W
HUBCD	:	Hub class sample driver
MGR	:	Peripheral device state manager of HCD
non-OS	:	USB basic firmware for OS less system
PP	:	Pre-processed definition
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function (non-OS)
Task	:	Processing unit
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB basic firmware for RZ/A1H Group

2. Module Configuration

The HHID comprises the HID class driver and device drivers for mouse and keyboard.
When data is received from the connected USB device, HCD notifies the application. Conversely, when the application issues a request, HCD notifies the USB device.
Figure 2-1 shows the structure of the HHID-related modules. Table 2.1 lists the modules and an overview of each.

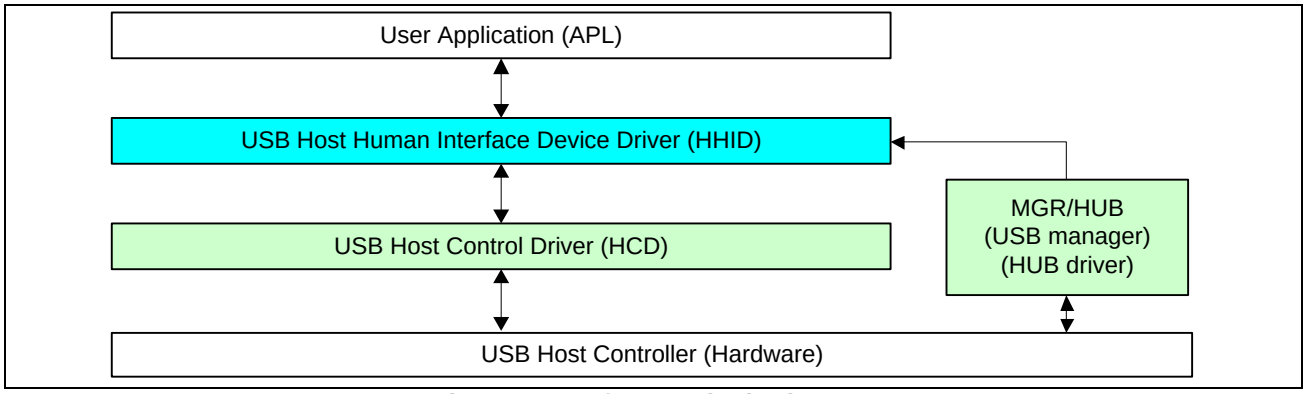

Figure 2-1 Software Block Diagram

Table 2.1 Module Function Descriptions

Module Name	Description
APL	User application program. The terminal software the information received from the HID device.
HHID	The HHID analyzes requests from HID devices. Notifies APL key operation information to the HID host via the HCD.
HCD/MGR	USB host Hardware Control Driver

3. System Resources

The resources used by HHID are listed below.

Table 3.1 Task information

Function Name	Task ID	Priority	Description
usb_hhid_task	USB_HHID_TSK	USB_PRI_3	HHID Task

Table 3.2 Mailbox information

Mailbox Name	Using Task ID	Task Queue	Description
USB_HHID_MBX	USB_HHID_TSK	FIFO order	For HHID

Table 3.3 Memory pool information

Memory Pool Name	Task Queue	Memory Block(Note)	Description
USB_HHID_MPL	FIFO order	40byte	For HHID

[Note]: The maximum number of memory blocks for the entire system is defined in USB_BLKMAX. The default value is 20.

4. Target Peripheral List (TPL)

When using a USB host driver (USB-BASIC-F/W) and device class driver in combination, it is necessary to create a target peripheral list (TPL) for each device driver.
For details on the TPL, refer to “How to Set a Targeted Peripheral List” in the Application Note of the USB Host and Peripheral Interface Driver (Document No: R01AN3291EJ).

5. Compile Setting

In order to use this module, it is necessary to set the USB-BASIC-F/W as a host. Refer to USB Basic Firmware application note (Document No. R01AN3291EJ) for information on USB-BASIC-F/W settings.
Please modify r_usb_hhid_config.h when User sets the module configuration option.
The following table shows the option name and the setting value.

Configuration options in r_usb_hhid_config.h	
USB_CFG_HHID_INT_IN USB_CFG_HHID_INT_IN2 USB_CFG_HHID_INT_IN3	Specifies the pipe number which is used at the data transfer. (Specifies any one from USB_PIPE6 to USB_PIPE8. Don't specify the same pipe number.)

6. Human Interface Device Class (HID)

6.1 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

6.2 Class Requests (Host to Device Requests)

Table 6.1 lists the class requests supported by the driver.

Table 6.1 HID Class Requests

Symbol	Request	Code	Description
a	USB_GET_REPORT	0x01	Receives a report from the HID device
b	USB_SET_REPORT	0x09	Sends a report to the HID device
c	USB_GET_IDLE	0x02	Receives a duration (time) from the HID device
d	USB_SET_IDLE	0x0A	Sends a duration (time) to the HID device
e	USB_GET_PROTOCOL	0x03	Reads a protocol from the HID device
f	USB_SET_PROTOCOL	0x0B	Sends a protocol to the HID device
	USB_GET_REPORT_DESCRIPTOR	Standard	Transmits report descriptor
	USB_GET_HID_DESCRIPTOR	Standard	Transmits an HID descriptor

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

a). GetReport Request Format

Table 6.2 shows the GetReport request format.

Receives a report from the device in a control transfer.

Table 6.2 GetReport Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_REPORT (0x01)	ReportType & ReportID	Interface	ReportLength	Report

b). SetReport Request Format

Table 6.3 shows the SetReport request format.

Sends report data to the device in a control transfer.

Table 6.3 SetReport Format

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

c). GetIdle Request Format

Table 6.4 shows the GetIdle request format.

Acquires the intarval time of the report notification (interrupt transfer). Idle rate is indicated in 4 ms units.

Table 6.4 GetIdle Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_IDLE (0x02)	0(Zero) & ReportID	Interface	1(one)	Idle rate

d). SetIdle Request Format

Table 6.5 shows the SetIdle request format.

Sets the interval time of the report notification (interrupt transfer). Duration time is indicated in 4 ms units.

Table 6.5 SetIdle Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_IDLE (0x0A)	Duration & ReportID	Interface	0(zero)	Not applicable

e). GetProtocol Request Format

Table 6.6 shows the GetProtocol request format.

Acquires current protocol (boot protocol or report protocol) settings.

Table 6.6 GetProtocol Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_PROTOCOL (0x03)	0(Zero)	Interface	1(one)	0(BootProtocol) / 1(ReportProtocol)

f). SetProtocol Request Format

Table 6.7 shows the SetProtocol request format.

Sets protocol (boot protocol or report protocol).

Table 6.7 SetProtocol Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_PROTOCOL (0x03)	0(BootProtocol) / 1(ReportProtocol)	Interface	0(zero)	Not applicable

6.2.1 Class Request Structure

The table below shows the structure used by the APIs *control()* and *R_usb_hhid_class_request()* to issue HHID specific requests.

Table 6.1 USB_HHID_CLASS_REQUEST_PARM_t structure

Type	Member name	Description
uint16_t	devadr	Device address
USB_REGADR_t	ipp	USB IP base address
uint16_t	ip	USB IP Number
uint16_t	bRequestCode	Class request code
void*	tranadr	Transfer data buffer
uint32_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

6.2.2 HID-Report Format

(1). Receive Report Format

Table 6.8 shows the receive report format used for notifications from the HID device.
Reports are received in interrupt IN transfers or class request GetReport.

Table 6.8 Receive Report Format

Offset	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 6.9 shows the format of the transmit report sent to the HID device.
Reports are sent in the class request SetReport.

Table 6.9 Transmit Report Format

Offset	Keyboard	Mouse
Data length	1 Byte	Not supported
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. USB Human Interface Device Class Driver (HHID)

7.1 Basic Functions

The HHID driver provides the following basic functions.

(1) Provides data transmit/receive services and a HID device.
(2) Provides HID class request services.

7.2 HHID API Functions

Table 7.1 shows the HHID driver API.

[Note]

If you want to use the API, which is described in USB Host and Peripheral Interface Driver (Document No: R01AN3291EJ), in the application program, you do not need to use the following API.

Table 7.1 HHID API Function List

Function	Description
R_usb_hhid_task	HHID Task
R_usb_hhid_DriverRelease	Releases the host HID class.
R_usb_hhid_TransferEnd	USB data transfer termination request
R_usb_hhid_DeviceInformation	Gets the HID device state information.
R_usb_hhid_ChangeDeviceState	Changes the device state.
R_usb_hhid_GetReportLength	Gets the report length.
R_usb_hhid_SetPipeRegistration	Sets the hardware pipe configuration.
R_usb_hhid_get_hid_protcol	Gets the protocol code
R_usb_hhid_driver_start	HHID driver start processing.
R_usb_hhid_class_request	Sends the class request
R_usb_hhid_PipeTransfer	USB data transfer reques
R_usb_hhid_class_check	Descriptor checking process
R_usb_hhid_get_pipetbl	Gets the pipe information address

7.2.1 R_usb_hhid_task

HHID Task

Format

void	R_usb_hhid_task(USB_VP_INT_t stacd)

Argument

stacd	Task Start Code (No used)

Return Value

—	—

Description

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

Note

1. Call this API in the user application program or the class driver.
2. This API is registered for scheduling by the task switching processing routine.

Example

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

    if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
    {
       R_usb_hstd_HcdTask((USB_VP_INT)0);        /* HCD Task */
       R_usb_hstd_MgrTask((USB_VP_INT)0);        /* MGR Task */
       R_usb_hhub_Task((USB_VP_INT)0);           /* HUB Task */
       R_usb_hhid_task((USB_VP_INT)0);           /* HHID Task */
       usb_hhid_main_task((USB_VP_INT)0);        /* HHID Application Task */
    }
    else
    {
       /* Idle Task (sleep sample) */
       R_usb_cstd_IdleTask(0);
    }
  }
}
```

7.2.2 R_usb_hhid_DriverRelease

Release the hid driver

Format

void R_usb_hhid_DriverRelease(USB_UTR_t *ptr)

Argument

*ptr Pointer to USB Transfer structure

Return Value

— —

Description

Release the registerd HHID class driver

Note

1. When the registered HHID is unnecessary, please call this function in the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number

Example

```
void usb_smp_task( void )
{
  USB_UTR_t *ptr;
    :
  R_usb_hhid_DriverRelease(ptr);
    :
}
```

7.2.3 R_usb_hhid_PipeTransfer

USB data transfer request

Format

USB_ER_t	R_usb_hhid_PipeTransfer (USB_UTR_t *ptr, uint8_t *buf, uint32_t size, USB_CB_t complete, uint16_t pipe)

Argument

*ptr	Pointer to USB Transfer structure
*buf	Pointer to the data buffer area
size	Read data size
complete	Call-back function
pipe	Pipe No.

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

This function requests a data transfer to the USB device.
When data transfer ends (specified data size reached, short packet received, error occurred), the call-back function is called.
Information on remaining transmit/receive data length, status, error count and transfer end is available in the parameter of the call-back function.

Note

1. Call this function from the user application program or class driver.
2. Specify the area other than the auto variable (stack) area to the 2nd argument.
3. When the received data is n times of the maximum packet size and less than the specified size in the argument (*size*), it is considered that the data transfer is not ended and a callback function (*complete*) is not generated.
4. Please set the following member of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
USB_ER_t usb_smp_task(void)
{
                        :
  usbip.ip = USB_HOST_USBIP_NUM;
  usbip.ipp = R_usb_cstd_GetUsbIpAdr( usbip.ip );
                        :
  R_usb_hhid_PipeTransfer(&usbip, buf, size, (USB_CB_t)usb_data_received);

}

/* Callback function */
void usb_data_received(USB_UTR_t *mess, uint16_t data2, uint16_t data3)
{
         :
}
```

7.2.4 R_usb_hhid_TransferEnd

USB data transfer termination request

Format

USB_ER_t R_usb_hhid_TransferEnd(USB_UTR_t *ptr, uint16_t pipe, uint16_t status)

Argument

*ptr	Pointer to USB Transfer structure
pipe	Pipe No.
status	USB communication status

Return Value

USB_E_OK	Success.
USB_E_ERROR	Failure

Description

This function forces data transfer via the pipes to end.
The function executes a data transfer forced end request to the HCD. After receiving the request, the HCD executes the data transfer forced end request processing.
When a data transfer is forcibly ended, the function calls the call-back function set in (R_usb_hhid_PipeTransfer) at the time the data transfer was requested. The remaining data length of transmission and reception, status, the number of times of a transmission error, and the information on forced termination are set to the argument (ptr) of this callback function

Note

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. Specify the area other than the auto variable (stack) area to the 1st argument.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t status;
  uint16_t pipe;
    :
  pipe   = USB_PIPE1;
  status = USB_DATA_STOP;

  /* Transfer end request */
  err = R_usb_hhid_TransferEnd(ptr, pipe, status);

  return err;
    :
}
```

7.2.5 R_usb_hhid_DeviceInformation

Get the HID device information

Format

void	R_usb_hhid_DeviceInformation(USB_UTR_t *ptr, uint16_t devaddr, uint16_t *tbl)

Argument

*ptr	Pointer to USB Transfer structure
devaddr	USB device address
*tbl	Pointer to the table address for device information storing

Return Value

—	—

Description

The information on the device connected to the USB port is acquired.
The information stored in a device information table is shown below.

- [0] Root port number to which device is connected
- [1] Device state
- [2] Configuration number
- [3] Interface class code 1
- [4] Connection speed
- [5] --
- [6] --
- [7] --
- [8] Status of rootport0
- [9] Status of rootport1

Note

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. This class driver does not support multiple interfaces, [5], [6] and [7] above are not used.
4. Use a 20-byte area for argument *tbl.

Example

```
void  usb_smp_task(void)
{
  USB_UTR_t  usbip;
  uint16_t   tbl[10];
    :
  usbip.ip = USB_HOST_USBIP_NUM;     /* Setting USB IP No */
  /* Confirm the device information */
  R_usb_hhid_DeviceInformation(&usbip, devaddr, &tbl);
    :
}
```

7.2.6 R_usb_hhid_ChangeDeviceState

Changes device state

Format

void R_usb_hhid_ChangeDeviceState(USB_UTR_t *ptr, uint16_t msginfo)

Argument

*ptr	Pointer to USB Transfer structure
msginfo	USB communication status

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

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. Specify the area other than the auto variable (stack) area to the 1st argument.

Example

```
void  usb_smp_task( void )
{
              :
  usbip.ip = USB_HOST_USBIP_NUM;
  usbip.ipp = R_usb_cstd_GetUsbIpAdr( USB_HOST_USBIP_NUM );
              :
  /* Change the device state request */
  R_usb_hhid_ChangeDeviceState(&usbip, USB_DO_GLOBAL_SUSPEND);
    :
}
```

7.2.7 R_usb_hhid_SetPipeRegistration

Set USB hardware pipe configuration

Format

void R_usb_hhid_SetPipeRegistration(USB_UTR_t *ptr, uint16_t devadr)

Argument

*ptr	Pointer to USB Transfer structure
devadr	USB device address

Return Valaue

—	—

Description

This function configures the hardware pipes. Each pipe is set according to the contents of the pipe information registered during HHID registration.

Note

1. Call this function from the user application program during initialization.
2. Please set the following member of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

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

7.2.8 R_usb_hhid_get_hid_protocol

Get the protocol code

Format

uint8_t R_usb_hhid_get_hid_protocol(uint16_t ipno, uint16_t devadr)

Argument

ipno USB module number
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

1. Call this function from the user application program or class driver.
2. bInterfaceProtocol is included in Interface Descriptor.
3. Specifies USB module number which HID device is connected to in the argument "*ipno*".

USB Module	USB Module Number
USB0	USB_IP0
USB1	USB_IP1

Example

```
void  usb_smp_task( void )
{
  uint8_t   protocol;
    :
  /* Gets the interface protocol value */
  protocol = R_usb_hhid_get_hid_protocol( USB_IP0, devadr );
    :
}
```

7.2.9 R_usb_hhid_driver_start

HHID driver start

Format

void	R_usb_hhid_driver_start(USB_UTR_t *ptr)

Argument

*ptr	Pointer to USB Transfer structure

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

1. Call this function from the user application program during initialization.
2. Please set the following member of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void usb_hstd_task_start( void )
{
  USB_UTR_t  *ptr;
    :
  ptr->ip = USB_HOST_USBIP_NUM;   /* USB IP No */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP base address */
    :
  R_usb_hhid_driver_start( ptr );    /* Host Class Driver Task Start Setting */
  usb_hstd_usbdriver_start( ptr );   /* Host USB Driver Start Setting */
  usb_hapl_registration( ptr );      /* Host Application Registration */
  usb_hapl_task_start( ptr );        /* Host Application Task Start Setting */
    :
}
```

7.2.10 R_usb_hhid_class_request

Send HID class request

Format

USB_ER_t R_usb_hhid_class_request(void *pram)

Argument

*pram HID class request structure. See 6.2.1 .

Return Value

USB_E_OK Success.
USB_E_ERROR Failure

Description

This function request HID class request issue to HID driver.

Note

1. Call this function from the user application program or class driver. Please refer to “Example”.
2. The class requests listed below can be called using this API. Please assign the desired Request Code to “bRequestCode” member in USB_HHID_REQUEST_PARAM_t structure before calling.

Class Request	Definition Value
Get_Descriptor(HID)	USB_HID_GET_HID_DESCRIPTOR
Get_Descriptor(Report)	USB_HID_GET_REPORT_DESCRIPTOR
Get_Descriptor(Physical)	USB_HID_GET_PHYSICAL_DESCRIPTOR
Set_Report	USB_HID_SET_REPORT
Get_Report	USB_HID_GET_REPORT
Set_Idle	USB_HID_SET_IDLE
Get_Idle	USB_HID_GET_IDLE
Set_Protocol	USB_HID_SET_PROTOCOL
Get_Protocol	USB_HID_GET_PROTOCOL

Example

```
void usb_hhid_smpl_set_report(USB_UTR_t *ptr, 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.ip = ptr->ip;
  class_req.ipp = ptr->ipp;
  class_req.tranadr = p_data;
  class_req.tranlen = length;
  class_req.complete = complete;

    R_usb_hhid_class_request((void*)&class_req);

}
```

7.2.11 R_usb_hhid_GetReportLength

Gets HID Report length

Format

uint16_t R_usb_hhid_GetReportLength(uint16_t ipno, uint16_t devadr)

Argument

ipno	USB module number
devadr	Device address

Return Value

—	Max packet size

Description

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

Note

1. Call this function from the user application program.
2. Specifies USB module number which HID device is connected to in the argument "*ipno*".

USB Module	USB Module Number
USB0	USB_IP0
USB1	USB_IP1

Example

```
void usb_smp_task( void )
{
  uint16_t   usb_smp_report_length;
    :
  usb_smp_report_length = R_usb_hhid_GetReportLength(USB_IP0, devadr);
    :
}
```

7.2.12 R_usb_hhid_class_check

Gets the descriptor information

Format

USB_ER_t R_usb_hhid_class_check(USB_UTR_t *ptr, uint16_t **table)

Argument

*ptr	Pointer to USB Transfer structure
**table	Pointer to the pipe information 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
[8] : Pipe information table

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 sets the descriptor check result (table[3]) to USB_DONE if the check result is OK and to USB_ERROR if the check result is NG.
The function acquires the descriptor information for the peripheral device.

Note

—

Example

```
void  usb_hhid_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t  driver;

  driver.ifclass      = (uint16_t)USB_IFCLS_HID;
        :
  driver.classcheck   = (USB_CB_CHECK_t)&R_usb_hhid_class_check;
        :
  driver.devresume = (USB_CB_INFO_t)&usb_hhid_dummy_function;
  R_usb_hstd_DriverRegistration(ptr, (USB_HCDREG_t*)&driver);
}
```

7.2.13 R_usb_hhid_get_pipetbl

Get pipe information table address

Format

uint16_t* R_usb_hhid_get_pipetbl (USB_UTR_t *ptr, uint16_t devadr)

Argument

*ptr	Pointer to USB Transfer structure
devadr	Device address

Return Value

—	—

Description

Gets the address of the pipe information table used for data communication with the device at the address passed by the second argument.

Note

1. Call this function from the user application program
2. Please set the following member of USB_UTR_t structure.
   - USB_REGADR_t ipp : USB register base address
   - uint16_t ip : USB IP Number
3. Specify the area other than the auto variable (stack) area to the 1st argument.

Example

```
void R_usb_hhid_SetPipeRegistration(USB_UTR_t *ptr, uint16_t devadr)
{
  uint16_t    *pipetbl;

  pipetbl = R_usb_hhid_get_pipetbl( ptr,  devadr);
  pipetbl[3] |= (uint16_t)(devadr << USB_DEVADDRBIT);
  R_usb_hstd_SetPipeRegistration(ptr, pipetbl, pipetbl[0]);

}
```

8. Sample Application

8.1 Application Specifications

Transfers data to and from an HID device (mouse or keyboard) connected to the GENMAI. Data received from the HID device is displayed on an terminal software.

[Note]

1. Up to three HID devices can be connected to a single USB module by using a USB hub.

8.2 Application Processing

The APL comprises two parts: initial setting and main loop. The following gives the processing summary for each part.

8.2.1 Initial Setting

In the initial setting part, the initial setting of the USB controller and the initialization of the application program are performed.

8.2.2 Main Loop

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after an HID device attaches to the USB host (GENMAI) and enumeration finishes, USB_STS_CONFIGURED is set as the return value. When the APL confirms USB_STS_CONFIGURED, it calls the R_USB_Write function to request transmission of data to the HID device.
2. When the R_USB_GetEvent function is called after sending of class request SET_PROTOCOL to the HID device has finished, USB_STS_REQUEST_COMPLETE is set as the return value. When the APL confirms USB_STS_REQUEST_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
3. When the R_USB_GetEvent function is called after reception of data from the HID device has finished, USB_STS_READ_COMPLETE is set as the return value. When APL is confirming the USB_STS_READ_COMPLETE, to display the data received from the HID device to the terminal software. Then, call the R_USB_Read function, it is possible to transmit and receive data is data reception request of data is from the HID device.
4. The processing in step 3, above, is repeated.

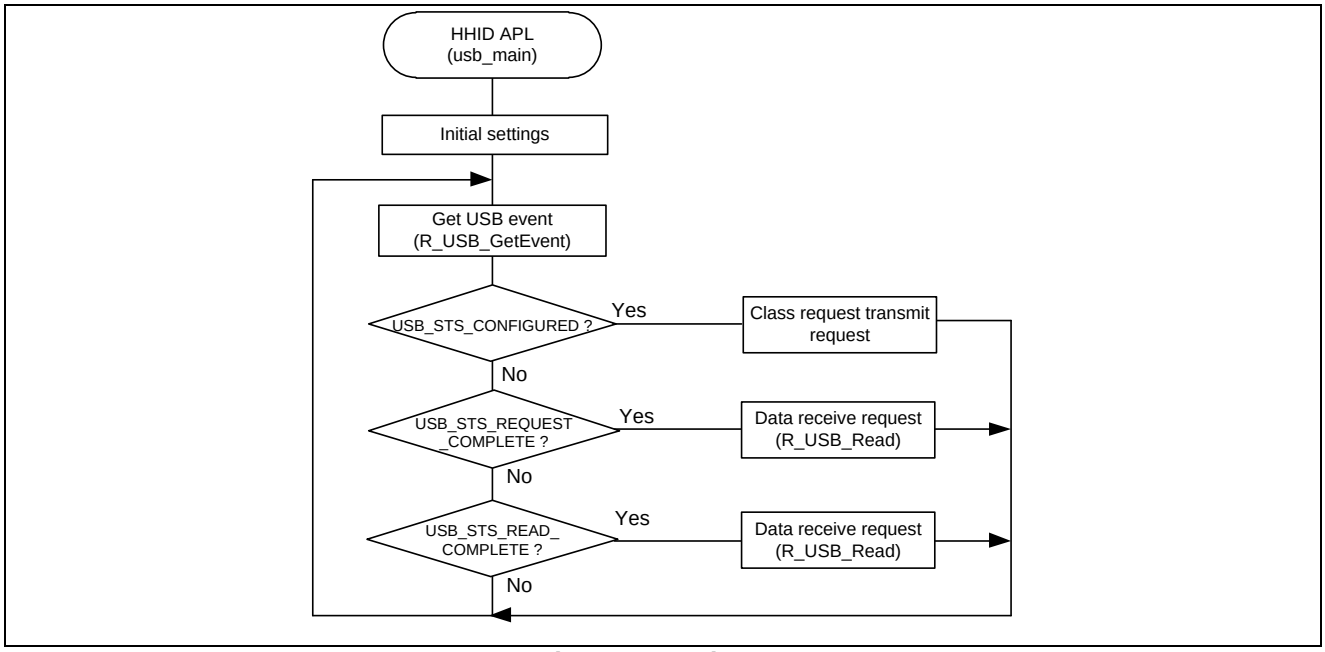

Figure 8-1 Main Loop

8.3 Display Information

The APL displays on the terminal software screen the connection state of the HID device and data received from the connected HID device.

Mouse connected : Displays on the LCD the amount of movement on the X and Y axes.
Keyboard connected : Displays on the LCD the last input key data.

The terminal software indication does not change when the data received from the HID device is NULL (no key on keyboard pressed, mouse not moved on X or Y axes).

9. Setup

9.1 Hardware

Figure 9-1 shows an example operating environment for the HMSC. Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

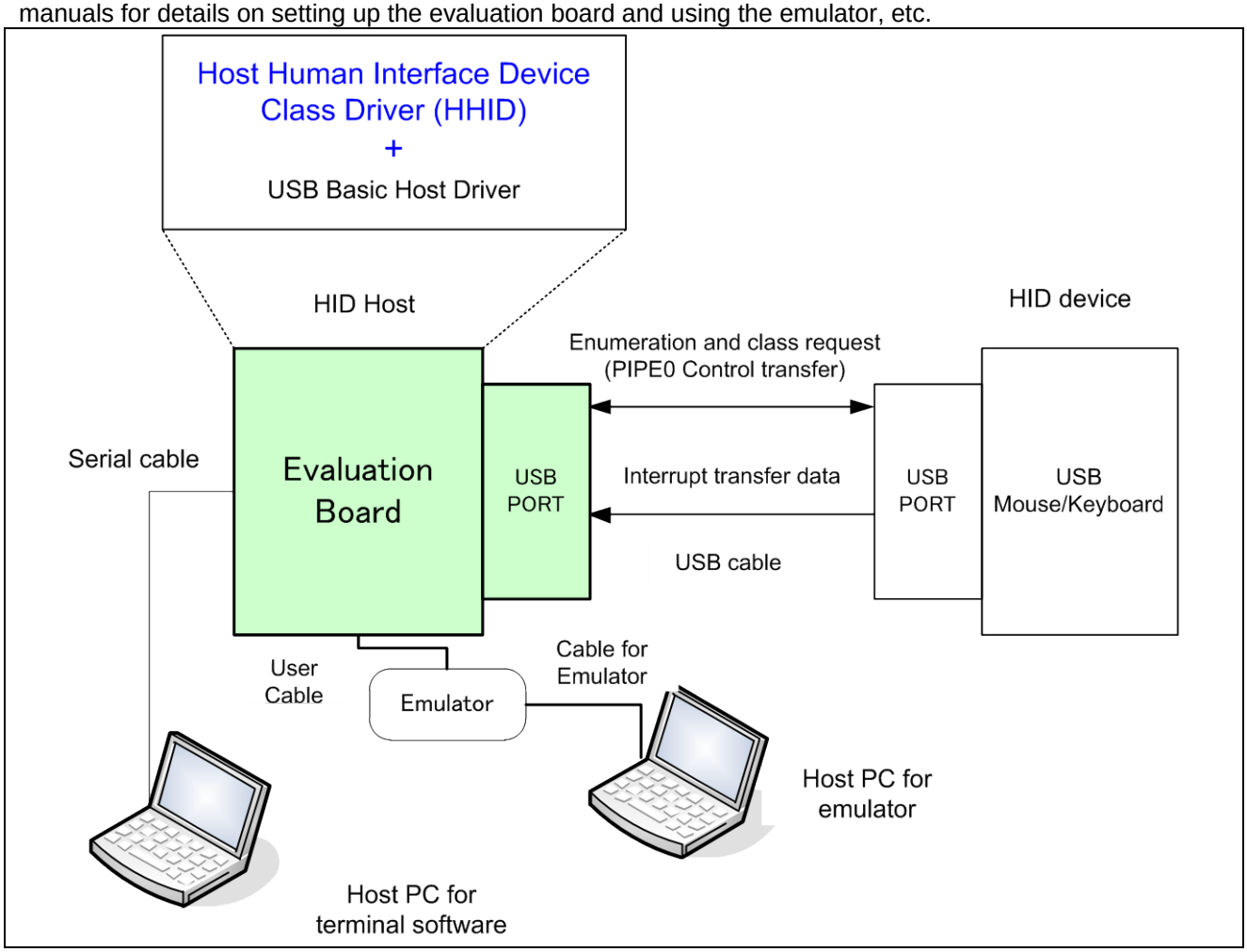

Figure 9-1 Example Operating Environment

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep 30, 2016	—	First edition issued

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.

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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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 systems; anti-disaster systems; anti-crime systems; and safety 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 (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable 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. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages 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 its 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 be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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. Because the evaluation of microcomputer software alone is very difficult, 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 use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(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.

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, HALII 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