RZ/A2M Group

USB Basic Host Driver

Introduction

This application note describes the basic USB firmware. This firmware controls hardware for USB communications. The firmware is hereafter referred to as the USB-BASIC-F/W.

Target Device

RZ/A2M Group

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

Contents

1. Overview

The USB-BASIC-F/W is used to control the USB 2.0 HS host module. The firmware runs in combination with sample device class drivers provided by Renesas or device class drivers created by the user.

The firmware supports the following functions.

- Connection and disconnection of devices, suspension and resumption, and USB bus reset processing
- Control transfer, bulk transfer, bulk transfer, interrupt transfer, and isochronous transfer
- Low-Speed/full-speed/high-speed function device and enumeration (operating speed differs according to the device)
- Detecting transfer errors and retrying transfer
- Multiple device class drivers can be mounted without the need to customize the USB-BASIC-F/W .

The names of API etc are different between port 0 and port 1.

In this document, API names etc. of port 0 are described as an example.

Table 1.1 Peripheral device used

Peripheral device	Usage
Host PC	output messages from the sample code

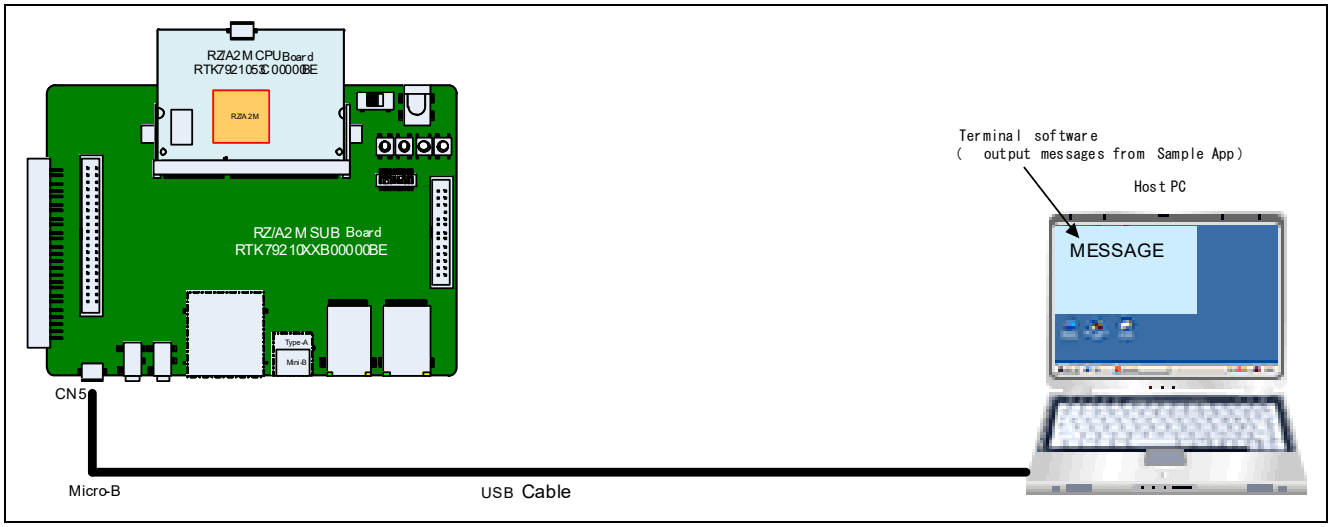

Figure 1.1 Operation check conditions

1.1 Note

When issuing class-specific and vendor-specific requests, adjusting the transfer rate, program capacity, etc., or individually setting the user interface, you will need to customize these settings.

The USB-BASIC-F/W firmware is not guaranteed for operations for USB communications. When utilizing it in a system, the customer should verify operation and also confirm its connection to various devices.

1.2 Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

Abbreviation	Full Form
APL	Application program
cstd	Prefix of function and file for USB-BASIC-F/W
HCD	Host control driver of USB-BASIC-F/W
HDCD	Host device class driver (device driver and USB class driver)
hstd	Prefix of function and file for Host USB-BASIC-F/W
HUBCD	Hub class sample driver
H/W	Renesas USB device RZ/A2M Group
MGR	Device state manager of HCD
USB	Universal Serial Bus
USB-BASIC-F/W	USB Basic Host firmware for RZ/A2M Group
Scheduler	Used for simple scheduling of operations for OSLess tasks
Scheduler macro	Used for calling a OSLess scheduler
Task	Unit for processing

2. Operation Confirmation Conditions

Table 2.1 Operation Confirmation Conditions(1/2)

item	Contents
Microcomputer used	RZ/A2M
Operating frequency (Note)	CPU Clock (Iϕ) : 528MHz Image processing clock (Gϕ) : 264MHz Internal Bus Clock (Bϕ) : 132MHz Peripheral Clock 1 (P1ϕ) : 66MHz Peripheral Clock 0 (P0ϕ) : 33MHz QSPI0_SPCLK : 66MHz CKIO : 132MHz
Operating voltage	Power supply voltage (I/O): 3.3 V Power supply voltage (either 1.8V or 3.3V I/O (PVcc SPI)) : 3.3V Power supply voltage (internal): 1.2 V
Integrated development environment	e2 studio V7.6.0
C compiler	"GNU Arm Embedded Tool chain 6.3.1" compiler options(except directory path) Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0

Note: The operating frequency used in clock mode 1 (Clock input of 24MHz from EXTAL pin)

Table 2.2 Operation Confirmation Conditions(2/2)

Operation mode	Boot mode 3 (Serial Flash boot 3.3V)
Terminal software communication settings	• Communication speed: 115200bps • Data length: 8 bits • Parity: None • Stop bits: 1 bit • Flow control: None
Board to be used	RZ/A2M CPU board RTK7921053C00000BE RZ/A2M SUB board RTK79210XXB00000BE
Device (functionality to be used on the board)	• Serial flash memory allocated to SPI multi-I/O bus space (channel 0) Manufacturer : Macronix Inc. Model Name : MX25L51245GXD • RL78/G1C (Convert between USB communication and serial communication to communicate with the host PC.) • LED1

3. Reference Application Notes

・None

4. Software Configuration

4.1 Module Configuration

The USB-BASIC-F/W comprises the host driver (HCD), host manager (MGR), hub-class driver (HUBCD). The HDCD is the class driver and not part of USB-BASIC-F/W .

The MGR manages the states of connected devices and performs enumeration. When the application (APL) changes the device state, call the corresponding API function from the APL to request a change of the state to the HCD or HUBCD.

The HUBCD is a sample program for enumerating and managing the states of devices connected to the downstream ports of a USB hub.

When issuing class-specific and vendor-specific requests, adjusting the transfer rate, program capacity, etc., or individually setting the user interface, you will need to customize these settings. Figure 4-1 is a block diagram of the USB-BASIC-F/W , and Table 4-1 gives a summary of the module functions.

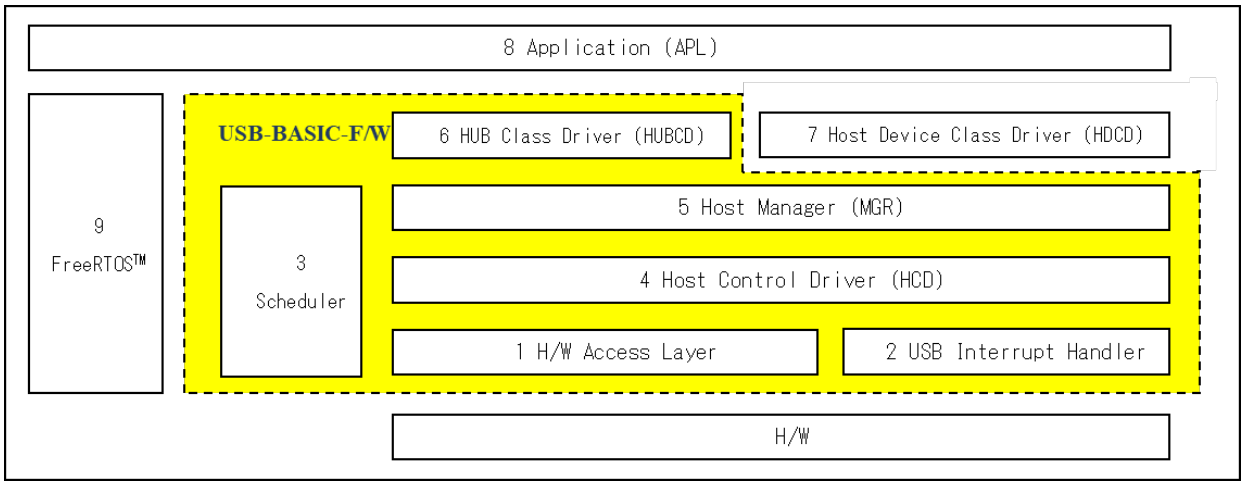

Figure 4-1 Module Configuration of the USB-BASIC-F/W

Table 4-1 Summary of the Module Functions

No	Module	Description
1	H/W Access Layer	USB2.0 HS host module control
2	USB Interrupt Handler	USB interrupt handler
3	Scheduler	Scheduler function
4	Host Control Driver	Host transaction management
5	Host Manager	Device state management, processing for enumeration HCD/HUBCD control message determination
6	HUB Class Driver	Management of the states of devices connected to the hub's downstream ports, processing for enumeration
7	Host Device Class Driver	Device class processing (prepared by the user to suit the system)
8	Application	Application (prepared by the user to suit the system)
9	FreeRTOS™	FreeRTOS™

4.2 Scheduler Function

This firmware uses a scheduler to manage requests generated by tasks and hardware according to their relative priority. When multiple task requests with the same priority are generated, a FIFO queue determines their order of execution.

Requests between the tasks are implemented by transmitting and receiving messages. In addition, callback functions are used for responses to tasks indicating the end of a request, so the user need only install appropriate class drivers for the system and there is no need to modify the scheduler itself.

For details of the schedule function, see section 7.

4.3 Host Device Class Control Driver (HDCD)

4.3.1 Registration

A HDCD suited to the user system must be created.

Register various information on the created HDCD in the st_usbh0_hcdreg_t structure by using the API function R_USBH0_HstdDriverRegistration().

For details of the method of registration, see section 5.6.6.

4.3.2 Setup of Tasks

When the HDCD uses a scheduler, see section 7, Scheduler.

For each additional task, add the task ID, mailbox ID, and memory pool ID to the r_usbh0_basic_config.h file.

When setting task priorities, use the API function R_USBH0_CstdSetTaskPri().

Note the following when setting these items.

- Assign consecutive ID numbers to tasks, and do not assign the same ID to more than one task.
- Set the same value as task ID for mailbox ID and memory pool ID.
- Set the priority of added tasks to a value lower than that of HCD(1), MGR(2), or HUB(3), with 0 being the highest and 7 being the lowest.

Examples of additional settings in r_usbh0_basic_config.h are listed below.

```
#define USB_SMP_TSK     USB_TID_6           :  Task ID
#define USB_SMP_MBX     USB_SMP_TSK         :  Mailbox ID
#define USB_SMP_MPL     USB_SMP_TSK         :  Memory pool ID
```

Set the following items in the r_usbh0_basic_config.h file as required.

- Maximum value of the task ID (see section 5.1.7)
- Maximum number of memory pools (see section 5.1.9)
- Maximum number of message pools (see section 5.1.10)

4.3.3 Class Checking

Use the class check callback function for class checking during enumeration.

On completion of processing, call the API function R_USBH0_HstdReturnEnuMGR() to convey the result.

4.4 Host Control Driver (HCD)

4.4.1 Basic Functions

The HCD is a program for controlling the hardware. The functions of the HCD are listed below.

1. Control transfer (ControlRead/ControlWrite/No-dataControl) and notification of results
2. Data transfer (Bulk/Interrupt/Isochronous) and notification of results
3. Forced end of data transfer (all pipes)
4. Error detection in USB communications and retrying transfer
5. Sending reset signals for the USB bus and notification of the results of the reset handshake
6. Transmission of suspend and resume signals
7. Generation of interrupts to indicate the detection of attachment and detachment

4.4.2 Issuing Requests to the HCD

API functions are used when hardware control requests are issued to the HCD.

For requests to update the state of the connected devices, control may be exercised by the HCD either directly or via a USB hub. The MGR determines the device address and issues control requests to the HCD and HUBCD tasks.

In response to requests from the higher layer task, the HCD sends notifications of the results by means of callback functions.

4.4.3 When Connecting Multiple Devices

Enumeration proceeds with the device address of the connected hub as 1. In addition, the HUBCD automatically assigns device addresses of 2 and higher to the devices connected to the downstream ports.

Enumeration and communications with two or more devices are possible, but the HUBCD cannot enumerate multiple devices simultaneously. In such cases, after the first connected device has been enumerated, enumeration of the next connected device starts, and so on.

Furthermore, the HUBCD does not calculate the USB bus occupancy ratios.

4.5 Host Manager (MGR)

4.5.1 Basic Functions

The MGR is a task that provides supplementary functionality between the HCD and HDCD. The functions of the MGR are listed below.

1. Registration of HDCD
2. State management for connected devices
3. Enumeration of connected devices
4. Searching for endpoint information from descriptors

4.5.2 USB Standard Requests

The MGR enumerates connected devices.

The USB standard requests issued by the MGR are listed below. The descriptor information fetched from a device is stored internally and can be registered as pipe information by using the API function.

- GET_DESCRIPTOR (Device Descriptor)
- SET_ADDRESS
- GET_DESCRIPTOR (Configuration Descriptor)
- SET_CONFIGURATION

4.5.3 Class Checking

The MGR notifies the HDCD of the information obtained during enumeration by using the GET_DESCRIPTOR request and checks whether the connected device is ready to operate.

Enumeration continues according to the response returned by the R_USBH0_HstdReturnEnuMGR() API function of the HDCD.

4.6 HUB Class Driver (HUBCD)

4.6.1 Basic Functions

The HUBCD manages the states of the downstream ports of a connected USB hub and provides supplementary functionality between the HCD and HDCD. The functions of the HUBCD are listed below.

1. Enumeration of the USB hub
2. State management for devices connected to the USB hub downstream ports
3. Enumeration of devices connected to the USB hub downstream ports

4.6.2 State Management for Downstream Ports

When the USB hub is connected, the HUBCD performs the following for all downstream ports and confirms the device connection state of each downstream port.

1) Enable port power (HubPortSetFeature: USB_HUB_PORT_POWER)
2) Initialize ports (HubPortClrFeature: USB_HUB_C_PORT_CONNECTION)
3) Acquire the port status (HubPortStatus: USB_HUB_PORT_CONNECTION)

4.6.3 Connecting Devices to Downstream Ports

When the HUBCD receives notification of a device being attached to a downstream port of the USB hub, it issues a USB reset signal request to the USB hub (HubPortSetFeature: USB_HUB_PORT_RESET).

After that, it uses the MGR task resource to enumerate the connected devices.

The HUBCD stores the results of the reset handshake and assigns successive unused device addresses to the connected devices.

4.6.4 Class Requests

Table 4-2 lists the class requests supported by the HUBCD.

Table 4-2 USB Hub Class Requests

Request	State of Installation	Function	Description
ClearHubFeature	×		
ClearPortFeature	✓	usbh0_hhub_port_clr_feature	USB_HUB_C_PORT_CONNECTION USB_HUB_C_PORT_RESET
ClearTTBuffer	×		
GetHubDescriptor	✓	usbh0_hhub_get_hub_descriptor	Acquires hub descriptor
GetHubStatus	×		
GetPortStatus	✓	usbh0_hhub_get_status	Acquires port status
ResetTT	×		
SetHubDescriptor	×		
SetHubFeature	×		
SetPortFeature	✓	usbh0_hhub_port_set_feature	USB_HUB_PORT_POWER USB_HUB_PORT_RESET
GetTTState	×		
StopTT	×		

5. Configurations

5.1 SmartConfigurator Configuration(r_usbh0_basic_drv_sc_cfg.h)

Specify the following definitions using SmartConfigurator.

5.1.1 OS/OSLess Select

Select OSLess / FreeRTOS.

```
#define         BSP_CFG_RTOS_USED   0       // OSLess
#define         BSP_CFG_RTOS_USED   1       // FreeRTOS
```

5.2 USB Host Basic Configurations(r_usbh0_basic_config.h)

Make settings for functionality of the USB-BASIC-F/W firmware by overwriting the common user-defined information file (r_usbh0_basic_config.h).

Change the values of the items below to suit the specifications of your system.

5.2.1 Device class setting

Enable the definition of the USB driver to be used among the definitions below.

Only one definition can be enabled.

```
#define   USB_CFG_HCDC_USE          // Host Communication Device Class
#define   USB_CFG_HHID_USE          // Host Human Interface Device Class
#define   USB_CFG_HMSC_USE          // Host Mass Storage Class
#define   USB_CFG_HVNDR_USE         // Host Vendor Class
```

5.2.2 Setting Maximum Number of Pipes

Set the maximum number of pipes.

One pipe is used for each endpoint of the connected device.

Setting a higher value increases the amount of memory used.

```
#define USBH0_MAXPIPE         32u
```

5.2.3 Setting Support for USB Embedded Host Compliance Test

Set support for a USB embedded host compliance test.

Define this macro when you want support for the compliance test.

```
   #define USBH0_HOST_COMPLIANCE_MODE  : Compliance test is supported
 //#define USBH0_HOST_COMPLIANCE_MODE  : Compliance test is not supported
```

5.2.4 Setting Compliance Test Information Display Function

Register the function for displaying information on a display device (LCD, etc.) when the compliance test is supported.

```
#define USBH0_COMPLIANCE_DISP(data1, data2) usb_cstd_DummyFunction(data1, data2)
```

5.2.5 Setting Function when Overcurrent is Detected

Register the function that is called in response to detection of overcurrent.

```
#define USBH0_OVERCURRENT(rootport)       usb_cstd_DummyFunction(rootport)
```

The local definition information file(r_usbh0_basic_local.h) has the following function settings.

5.2.6 Setting Maximum Device Address

Set the maximum device address (the maximum number of connectable devices).

Setting a higher value increases the amount of memory used.

```
#define USBH0_MAXDEVADDR      12u
```

5.2.7 Setting Maximum Task ID

Set the maximum number of tasks (maximum task ID) for the scheduler.

Setting a higher value increases the amount of memory used.

```
#define USB_IDMAX           11u
```

5.2.8 Setting Maximum Priority

Set the maximum priority for the scheduler.

Setting a higher value increases the amount of memory used.

```
#define USBH0_PRIMAX          8u
```

5.2.9 Setting Maximum Number of Memory Pools

Set the maximum number of memory pools for the scheduler.

Setting a higher value increases the amount of memory used.

```
#define USBH0_BLKMAX          21u
```

5.2.10 Setting Maximum Number of Message Pools

Set the maximum number of message pools for the scheduler.

Setting a higher value increases the amount of memory used.

```
#define USB_TABLEMAX        USBH0_BLKMAX
```

5.2.11 Setting Output of Debugging Information

Set output of debugging information.

```
   #define USBH0_DEBUG_OUTPUT   : Output of debugging information is enabled
 //#define USBH0_DEBUG_OUTPUT   : Output of debugging information is disabled
```

5.2.12 Setting Debugging Information Output Function

These macro definitions are for the output of debugging information through a UART, on a display device, etc. A serial driver or display device driver will also be required.

Register the functions to suit the user system.

```
#define USBH0_PRINTF0(FORM)                         printf(FORM)
#define USBH0_PRINTF1(FORM,x1)                      printf (FORM,x1)
#define USBH0_PRINTF2(FORM,x1,x2)                   printf (FORM,x1,x2)
#define USBH0_PRINTF3(FORM,x1,x2,x3)                printf (FORM,x1,x2,x3)
#define USBH0_PRINTF4(FORM,x1,x2,x3,x4)             printf (FORM,x1,x2,x3,x4)
#define USBH0_PRINTF5(FORM,x1,x2,x3,x4,x5)          printf (FORM,x1,x2,x3,x4,x5)
#define USBH0_PRINTF6(FORM,x1,x2,x3,x4,x5,x6)       printf (FORM,x1,x2,x3,x4,x5,x6)
#define USBH0_PRINTF7(FORM,x1,x2,x3,x4,x5,x6,x7)    printf (FORM,x1,x2,x3,x4,x5,x6,x7)
#define USBH0_PRINTF8(FORM,x1,x2,x3,x4,x5,x6,x7,x8)     printf
(FORM,x1,x2,x3,x4,x5,x6,x7,x8)
```

5.3 EHCI User-Defined Information File (r_usbh0_hehci_def_usr.h)

5.3.1 Specifying EHCI Periodic Frame List Size

Specify a value of 256, 512, or 1024 as the EHCI periodic frame list size.

This scheduling amplitude for periodic transfers changes according to the size specified.

Example) Specifying 256 bytes

```
#define USBH0_EHCI_PFL_SIZE  256
```

5.3.2 Specifying EHCI Queue Head Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI queue head data structure.

The queue head data structure must be able to accommodate the number of endpoint pipes used for control transfer, bulk transfer, and interrupt transfer.

Set a value that suits the specifications of your system.

Example) Specifying a value of 16

```
#define USBH0_EHCI_NUM_QH    16
```

5.3.3 Specifying EHCI qTD Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI qTD (device status register queue element transfer descriptor) data structure.

The qTD data structure is used as a transfer management descriptor linked to the queue head in control transfer, bulk transfer, and interrupt transfer.

The maximum data size that can be transferred with a single qTD is 20,480 bytes.

In addition, control transfer requires one qTD for each setup stage, data stage, and status stage.

Set a value that suits the specifications of your system.

Example) Specifying a value of 256

```
#define USBH0_EHCI_NUM_QTD   256
```

5.3.4 Specifying EHCI iTD Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI iTD (high-speed isochronous transfer descriptor) data structure.

The iTD data structure must be able to accommodate the number of endpoint pipes used for high-speed isochronous transfer.

Set a value that suits the specifications of your system.

Example) Setting a value of 4

```
#define USBH0_EHCI_NUM_ITD   4
```

5.3.5 Specifying EHCI siTD Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI siTD (split transaction isochronous transfer descriptor) data structure.

The siTD data structure must be able to accommodate the number of endpoint pipes used for split transaction isochronous transfer.

Set a value that suits the specifications of your system.

Example) Specifying a value of 4

```
#define USBH0_EHCI_NUM_SITD  4
```

5.3.6 Specifying EHCI iTD Data Structure Maximum Data Transfer Size

Specify the maximum data transfer size for the EHCI iTD data structure.

The transfer size value specifies the maximum transfer size for a single high-speed isochronous transfer (one transaction).

The supported maximum data size setting is 1024.

Set a value that suits the specifications of your system.

Example) Specifying 512 bytes

```
#define USBH0_EHCI_ITD_DATA_SIZE    512
```

5.3.7 Specifying EHCI siTD Data Structure Maximum Data Transfer Size

The maximum data transfer size for the EHCI siTD data structure is fixed to 184.

5.3.8 Specifying EHCI Timeout Time

Specify the EHCI timeout time in msec units.

Example) Specifying 3 seconds

```
#define USBH0_EHCI_TIMEOUT   3000
```

5.4 OHCI User-Defined Information File (r_usbh0_hohci_def_usr.h)

5.4.1 Specifying OHCI Endpoint Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI endpoint data structure.

The OHCI endpoint data structure must be able to accommodate the number of endpoint pipes used for control transfer, bulk transfer, interrupt transfer, and isochronous transfer.

Set a value that suits the specifications of your system.

Example) Specifying a value of 16

```
#define USBH0_OHCI_NUM_ENDPOINT     16
```

5.4.2 Specifying OHCI Endpoint Descriptor Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI endpoint descriptor data structure.

The OHCI endpoint descriptor data structure must be able to accommodate the number of endpoint pipes used for control transfer, bulk transfer, interrupt transfer, and isochronous transfer, plus 31 more for interrupt transfer scheduling.

Set a value that suits the specifications of your system.

Example) Specifying a value of 64

```
#define USBH0_OHCI_NUM_ED    64
```

5.4.3 Specifying OHCI Transfer Descriptor Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI transfer descriptor data structure.

The OHCI transfer descriptor data structure is used as a transfer management descriptor in control transfer, bulk transfer, interrupt transfer, and isochronous transfer.

The maximum data size that can be transferred with a single transfer descriptor is 8,192 bytes.

Set a value that suits the specifications of your system.

Example) Specifying a value of 256

```
#define USBH0_OHCI_NUM_TD    256
```

5.4.4 Specifying Maximum Number of OHCI Isochronous Devices

Specify the maximum number of OHCI isochronous devices.

Set a value that suits the specifications of your system.

Example) Specifying a value of 4

```
#define USBH0_OHCI_ISO_MAXDEVICE    4
```

5.4.5 Specifying Maximum OHCI Isochronous Data Transfer Size

Specify the maximum data transfer size for OHCI isochronous devices.

The transfer size value specifies the maximum transfer size for a single OHCI isochronous transfer (one transaction).

The supported maximum data size setting is 1023.

Set a value that suits the specifications of your system.

Example) Specifying 256 bytes

```
#define USB_OHCI_ISO_MAX_PACKET_SIZE       256
```

5.4.6 Specifying OHCI Timeout Time

Specify the OHCI timeout time in msec units.

Example) Setting 3 seconds

```
#define USBH0_OHCI_TIMEOUT   3000
```

5.5 Target Peripheral List (TPL)

Which USB devices the USB-BASIC firmware supports is specifiable.

Specify the supported USB devices in sets as the product IDs and vendor IDs of the USB devices in the target peripheral list (TPL).

If you need not specify supported USB devices, set USB_NOVENDOR and USB_NOPRODUCT for the vendor ID and product ID, respectively.

An example of how to create a TPL is shown below.

```
const uint16_t usbh0_hmsc_devicetpl[] =
{
   2,                  /* Number of list */
   0,                  /* Reserved */
   0xXXXX,             /* Vendor ID */
   0xXXXX,             /* Product ID */
};
```

[Note]

1. Two TPLs for the application program and for the USB hub (r_usbh0_hhubsys.c) are prepared. When support for a compliance test is selected, do not set USB_NOVENDOR and USB_NOPRODUCT in these two TPLs (refer to section 5.1.3 in section 5.1, Common User-Defined Information File, for the setting of support for the compliance test).
2. When support for a compliance test is selected, set the vendor ID and product ID of the USB hub used for the compliance test in the TPL for the USB hub.

5.6 Structure Definitions

This section describes the structures. The structures are defined in the r_usbh0_basic_if.h file.

5.6.1 st_usbh0_hcdreg_t Structure

st_usbh0_hcdreg_t is used to register HDCD information.

The callback function registered in st_usbh0_hcdreg_t is executed when the device state changes, etc.

Table 5-1 lists the members of the st_usbh0_hcdreg_t structure.

Table 5-1 Members of st_usbh0_hcdreg_t Structure

Type	Member	Description
uint16_t	rootport	The connected port number is registered.
uint16_t	devaddr	The device address is registered.
uint16_t	devstate	The connection state of the device is registered.
uint16_t	ifclass	Register the interface class code for HDCD.
uint16_t	*tpl	Register the target peripheral list for HDCD operation. See section 5.4, Target Peripheral List (TPL).
usbh0_cb_check_t	classcheck	Register the function to be started when checking HDCD. It is called when TPLs match.
usbh0_cb_t	devconfig	Register the function to be started for transition to the configured state. It is called when the SET_CONFIGURATION request is completed
usbh0_cb_t	devdetach	Register the function to be started for transition to the detached state.
usbh0_cb_t	devsuspend	Register the function to be started for transition to the suspended state.
usbh0_cb_t	devresume	Register the function to be started for transition to the resumed state.

5.6.2 st_usbh0_setup_t Structure

The st_usbh0_setup_t structure is a data structure of the USB request.

Table 5-2 lists the members of the st_usbh0_setup_t structure.

Table 5-2 Members of the st_usbh0_setup_t Structure

Type	Member	Description
uint16_t	type	Set bRequest[b15-b8] and bmRequestType[b7-b0].
uint16_t	value	Set wValue.
uint16_t	index	Set wIndex.
uint16_t	length	Set wLength.
uint16_t	devaddr	Device address

5.6.3 usbh0_utr Structure

This structure is used for USB communications and task messages.

Setting this as an argument of R_USBH0_HstdTransferStart() allows USB communications with the connected device.

Table 5-3 lists the members of the usbh0_utr structure.

Table 5-3 Members of the usbh0_utr Structure

Type	Member	Description
uint16_t	msginfo	Set message information which USB-BASIC-F/W tasks use. Setting is not required for USB transfer.
uint16_t	keyword	Set the pipe number for USB transfer.
uint16_t	result	USB transfer results are stored. See section 6.3 Notification of Transfer Results.
usbh0_utr_cb_t	complete	Set the address of the function to be executed when USB transfer ends. See section 5.7, Callback Functions.
void	*tranadr	Set the USB communication buffer address. • Reception or control read transfer Specifies the buffer address for storing received data. • Transmission or control write transfer Specifies the buffer address where data for transmission is stored. ・No data control transfer Ignored even if specified.
uint32_t	tranlen	Set the USB communication data length. • Reception or control read transfer Specifies the length of received data. After the completion of USB transfer, the value is updated to a value minus the length of the actually received data. • Transmission or control write transfer Specifies the length of transmission data. ・No data control transfer Specify “0”.
st_usbh0_setup_t	*setup	Specify the setup packet information for control transfer. See section 5.5.2, st_usbh0_setup_t Structure, and section 6.6, Control Transfer.

5.7 API Functions

The MGR, HUBCD and HDCD issue hardware control requests by using the API functions.

ANSI C99 is used. These types are defined in stdint.h.

This driver API follows the Renesas API naming standards.

All API calls and interface definitions supporting these calls are given in r_usbh0_basic_if.h.

Table 5-4 lists the API functions.

Table 5-4 API Functions

Function	port	Description
R_USBH0_HstdTransferStart R_USBH1_HstdTransferStart	0 1	Data transfer request
R_USBH0_HstdTransferEnd R_USBH1_HstdTransferEnd	0 1	Data transfer forced end request
R_USBH0_HstdDriverRegistration R_USBH1_HstdDriverRegistration	0 1	HDCD registration
R_USBH0_HstdReturnEnuMGR R_USBH1_HstdReturnEnuMGR	0 1	Indicates the completion of class checking
R_USBH0_HstdChangeDeviceState R_USBH1_HstdChangeDeviceState	0 1	Request for changing the connected device state
R_USBH0_HstdSetPipe R_USBH1_HstdSetPipe	0 1	Sets pipe information
R_USBH0_HstdGetPipeID R_USBH1_HstdGetPipeID	0 1	Acquires the pipe number
R_USBH0_HstdClearPipe R_USBH1_HstdClearPipe	0 1	Clears the pipe information
R_USBH0_HstdMgrOpen R_USBH1_HstdMgrOpen	0 1	Starts MGR task
R_USBH0_HstdMgrClose R_USBH1_HstdMgrClose	0 1	Stops MGR task
R_USBH0_HstdMgrTask R_USBH1_HstdMgrTask	0 1	MGR task
R_USBH0_HhubTask R_USBH1_HhubTask	0 1	HUB task

5.7.2 R_USBH0_HstdTransferStart

Data transfer request

Format

usbh0_er_t R_USBH0_HstdTransferStart(usbh0_utr * utr)

Argument

*utr USB transfer structure

Return value

USBH0_OK Success

USBH0_ERROR Failure

Description

This function requests data transfer based on the transfer information stored in utr.
When the data transfer ends (the specified data size reached, a short packet received, an error occurred), the callback function is called. The information of the remaining transmit/receive data length, status, and transfer end is set in the argument to this callback function.

Note

1. Call this function from the user application or class driver.
2. Set the following members of the structure as arguments: pipe number, start address of transfer data, transfer data length, and the callback function to call at the end of transfer.
3. If the device is being enumerated, this function returns USBH0_ERROR until its enumeration is completed.

Example

```
usbh0_utr    usb_smp_trn_Msg[USBH0_MAXPIPE_NUM + 1];

usbh0_er_t   usb_smp_task(usbh0_utr_cb_t complete, uint16_t pipe, uint32_t size,
uint8_t *table)
{
   :
  /* Set transfer information */
  trn_msg[pipe].keyword    = pipe;      /* Pipe no. */
  trn_msg[pipe].tranadr    = table;     /* Pointer to data buffer */
  trn_msg[pipe].tranlen    = size;      /* Transfer size */
  trn_msg[pipe].complete   = complete;  /* Callback function */

  /* Transfer start request */
  err = R_USBH0_HstdTransferStart(&trn_msg[pipe]);

  return err;
   :
}
```

5.7.4 R_USBH0_HstdTransferEnd

Data transfer forced end request

Format

usbh0_er_t	R_USBH0_HstdTransferEnd(uint16_t pipe_id)

Argument

pipe_id	Pipe number

Return value

USBH0_OK	Success
USBH0_ERROR	Failure

Description

This function issues a request to forcibly end data transfer via the specified pipes.

Remark

1. Call this function from the user application or class driver.

Example

```
void   usb_smp_task(void)
{
  uint16_t pipe;
    :
  pipe    = USB_PIPEx

  /* Transfer end request */
  err = R_USBH0_HstdTransferEnd(pipe);

  return err;
    :
}
```

5.7.6 R_USBH0_HstdDriverRegistration

Host device class driver (HDCD) registration

Format

void	R_USBH0_HstdDriverRegistration(st_usbh0_hcdreg_t *callback)

Argument

*callback	Class driver structure

Return value

—	—

Description

This function registers the HDCD information, which is registered in the class driver structure, in the HCD. It then updates the number of registered drivers controlled by the HCD, and registers the HDCD in a new area. The initialization callback function is executed following the completion of registration.

Note

1. Call this function from the user application program or class driver at the initial setup.
2. See Table 5-1 Members of st_usbh0_hcdreg_t Structure, for details of registration information.

Example

```
void   usb_smp_registration(void)
{
    st_usbh0_hcdreg_t    driver;

    /* Interface class */
    driver.ifclass      = (uint16_t)USB_IFCLS_VEN;
    /* Target peripheral list */
    driver.tpl          = (uint16_t*)&usb_gapl_devicetpl;
    /* Driver check */
    driver.classcheck   = &usb_hvendor_class_check;
    /* Device configured */
    driver.devconfig    = &usb_hvendor_apl_init;
    /* Device detach */
    driver.devdetach    = &usb_hvendor_apl_close;
    /* Device suspend */
    driver.devsuspend   = &usb_hvendor_apl_suspend;
    /* Device resume */
    driver.devresume    = &usb_hvendor_apl_resume;

    /* Driver registration */
    R_USBH0_HstdDriverRegistration(&driver);
}
```

5.7.8 R_USBH0_HstdReturnEnuMGR

Indicating the completion of class checking

Format

void	R_USBH0_HstdReturnEnuMGR(uint16_t cls_result)

Argument

cls_result	Class check result

Return value

—	—

Description

This function requests continuation of enumeration processing. The MGT analyzes the argument of the function and determines whether or not to initiate a transition of the connected USB device to the configured state.

Specify the result of class checking (USBH0_OK or USBH0_ERROR) as the argument (cls_result).

Note

1. Call this function at the time of class check processing.

Example

```
void   usb_smp_task(void)
{
    :
  (Enumeration processing)
    :
  R_USBH0_HstdReturnEnuMGR(USBH0_OK);
}
```

5.7.10 R_USBH0_HstdChangeDeviceState

Request for changing the connected device state

Format

usbh0_er_t R_USBH0_HstdChangeDeviceState(usbh0_utr_cb_t complete,
int16_t msginfo, uint16_t member)

Argument

complete	Callback function
msginfo	Message information
member	Device address/pipe number

Return value

USBH0_OK	Success
USBH0_ERROR	Failure

Description

Setting the following values in msginfo and calling this API lead to sending a request to the HCD to change the state of the connected USB device.

msginfo	Outline
USBH0_DO_GLOBAL_SUSPEND	Request for transition to the suspended state with remote wakeup enabled (request the device specific suspended state for devices downstream from the hub)
USBH0_DO_GLOBAL_RESUME	Global resume execution request (resume operation of the connected devices downstream from the ports)
USBH0_DO_CLR_STALL	STALL clear request

Note

1. Call this function from the user application program or class driver.
2. When setting USBH0_DO_CLR_STALL as the argument msginfo, set the pipe number as the argument member.

Example

```
void  usb_smp_task(void)
{
    R_USBH0_HstdChangeDeviceState(&usb_smp_callback, USBH0_DO_GLOBAL_SUSPEND, devaddr);
}
```

5.7.12 R_USBH0_HstdSetPipe

Setting pipe information

Format

usbh0_er_t R_USBH0_HstdSetPipe(uint16_t **table)

Argument

**table Class check information table pointer (see Table 5-6)

Return value

USBH0_OK Success

USBH0_ERROR Failure

Description

This function analyzes the received descriptor and sets pipe information.

Note

1. Call this function at the time of class check processing.
2. When the pipe information area is full, this function returns USBH0_ERROR.

Example

```
void   usb_smp_classcheck(uint16_t **table)
{
    R_USBH0_HstdSetPipe(table);
}
```

5.7.14 R_USBH0_HstdGetPipeID

Acquiring pipe number

Format

uint16_t R_USBH0_HstdGetPipeID(uint16_t devaddr, uint8_t type,
uint8_t direction, uint8_t ifnum)

Argument

devaddr	Device address
type	Endpoint attribute (USBH0_EP_ISO / USBH0_EP_BULK / USBH0_EP_INT)
direction	Endpoint direction (USBH0_EP_IN / USBH0_EP_OUT)
ifnum	Interface number

Return value

uint16_t	Pipe number

Description

This function returns the pipe number that matches the condition specified by the argument.

Note

1. Call this function from the user application program or class driver.
2. Call the function after setting pipe information using R_USBH0_HstdSetPipe().
3. When 0xFF is specified as the interface number, the interface number will be ignored when searching.

Example

```
void usb_smp_configured(uint16_t devaddr)
{
    usb_gsmp_pipe = R_USBH0_HstdGetPipeID(devaddr, USBH0_EP_BULK, USBH0_EP_IN, 0);
}
```

5.7.16 R_USBH0_HstdClearPipe

Clearing pipe information

Format

usbh0_er_t R_USBH0_HstdClearPipe(uint16_t devaddr)

Argument

devaddr Device address

Return value

USBH0_OK Success

Description

This function clears the pipe setting of the device address specified by the argument to release the area.

Note

1. Call this function from the user application program or class driver.

Example

```
void usb_smp_detach(uint16_t devaddr)
{
    R_USBH0_HstdClearPipe(devaddr);
}
```

5.7.18 R_USBH0_HstdMgrOpen

Starting MGR task

Format

usbh0_er_t R_USBH0_HstdMgrOpen(void)

Argument

— —

Return value

USBH0_OK Normal end

Description

This function initialize the registration state of MGR.

When BSP_CFG_RTOS_USED = 1, a semaphore for port 0 is created.

The value returned is always USBH0_OK.

Note

1. Call this function from the user application program at the initial setup.
2. Do not call this function after starting the MGR task.

Example

```
void   usb_smp_task(void)
{
    R_USBH0_HstdMgrOpen();
}
```

5.7.19 R_USBH0_HstdMgrClose

Stopping MGR task

Format

void	R_USBH0_HstdMgrClose(void)

Argument

—	—

Return value

—	—

Description

When BSP_CFG_RTOS_USED = 1, deletes the semaphore for port 0.

Note

1. After activating MGR task, call this API with user application program.
2. Please do not call this API before starting MGR task.

Example

```
void   usb_smp_task(void)
{
    R_USBH0_HstdMgrOpen();
    :
    R_USBH0_HstdMgrClose();
}
```

5.7.21 R_USBH0_HstdMgrTask

MGR task

Format

void	R_USBH0_HstdMgrTask(void)

Argument

—	—

Return value

—	—

Description

This function provides supplementary functionality between the HCD and HDCD when host controller operation is selected.

Note

1. Call this function in the loop that executes sheduler processing.

Example

```
void   usb_smp_mainloop(void)
{
    while(1)
    {
        /* Start scheduler */
        R_USBH0_CstdScheduler();
        /* Check flag */
        if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
        {
            R_USBH0_HstdMgrTask();
        }
    }
}
```

5.7.23 R_USBH0_HhubTask

HUB task

Format

void	R_USBH0_HhubTask(void)

Argument

—	—

Return value

—	—

Description

This function manages the states of the downstream ports of a connected USB hub and provides supplementary functionality between the HCD and HDCD.

Note

1. Call this function in the loop that executes scheduler processing.

Example

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Start scheduler */
  R_USBH0_CstdScheduler();
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    R_USBH0_HhubTask();
       :
  }
}
```

5.8 Callback Functions

Using callback functions eliminates the need to poll for events.

For example, when a user application requests data transfer by the HCD, a callback function is registered as a way of notifying the application of the result of data transfer to or from the device.

The tables below list the callback functions.

Table 5-5 R_USBH0_HstdTransferStart Callback

Name	Data transfer request callback function		
Call format	void (*usbh0_utr_cb_t)(st_usbh0_utr_t *)		
Argument	st_usbh0_utr_t*	*p_utr	st_usbh0_utr_t pointer to the argument of R_USBH0_HstdTransferStart()
Return value	—	—	—
Description	This function is called at the end of data transfer (i.e., the end of transfer of data of the size specified by the application, or the end of transfer when a short packet is received, etc.).		
Notes			

Table 5-6 classcheck Callback

Name	classcheck callback function		
Call format	void (*usbh0_cb_check_t)(uint16_t **)		
Argument	uint16_t	**table	table[0] descriptor start address table[1] configuration descriptor start address table[2] interface descriptor start address table[3] descriptor check result table[4] hub specification table[5] port number (not used) table[6] transfer rate (not used) table[7] device address
Return value	—	—	—
Description	When the class code registered in the registration (the member ifclass of the st_usbh0_hcdreg_t structure) matches with the class code of the received descriptor, the function registered in the member classcheck of the st_usbh0_hcdreg_t structure is called. table[3]: Check result Respond to indicate whether the HDCD is ready to operate by returning either of the following: USBH0_OK: HDCD is ready to operate. USBH0_ERROR: HDCD is not ready to operate. table[4]: Hub specification (for HUB driver) USBH0_FSHUB: for Full-Speed HUB USBH0_HSHUBS: for Hi-Speed HUB (Single) USBH0_HSHUBM: for Hi-Speed HUB (Multi) table[7]: Device address Device address which is assigned by the host at the set address		
Notes			

Table 5-7 State Change Callback

Name	State change callback function		
Call format	void (*usbh0_cb_t)(uint16_t)		
Argument	uint16_t	devaddr	Address of the connected device
Return value	—	—	—
Description	When a state change occurs, the function that was registered in the member of the st_usbh0_hcdreg_t structure in the registration is called. devconfig: Executed when the Set_Configuration request is issued. devdetach: Executed when detachment is detected. devsuspend: Executed at transition to the suspended state is devresume: Executed at transition to the resumed state.		
Notes			

6. USB Communications

6.1 Pipes

A pipe is a logical connection to an endpoint on a USB device.

A default pipe for control transfer (pipe number 0) is set in the USB-BASIC-F/W firmware.

For data transfer, pipe information must be set from the descriptors of the USB device by calling the API function R_USBH0_HstdSetPipe() at the time of class checking. Pipe information is allocated sequentially from pipe number 1 for each endpoint descriptor.

The maximum number of pipes can be changed by editing the config file. See section 5.1.2 for details.

The information to be set for the pipes is as follows.

- Interface number
- Endpoint number
- Endpoint transfer type
- Endpoint direction
- Max. packet size
- Device address

The pipe number must be specified for a data transfer request.

The pipe number can be acquired by calling the API function R_USBH0_HstdGetPipeID().

When the device has been disconnected, the pipe information must be cleared to release the pipe information area.

The pipe information can be cleared by calling the API function R_USBH0_HstdClearPipe().

6.2 Data Transfer Request

A data transfer request can be issued by transferring the usbh0_utr structure in which transfer information has been set as an argument of the API function R_USBH0_HstdTransferStart().

For details of the usbh0_utr structure, see section 5.5.3, usbh0_utr Structure.

6.3 Notification of Transfer Results

On completion of data transfer, the USB-BASIC-F/W notifies the HDCD of the completion of data transfer by using the callback function that was registered when data transfer was requested.

The results of transfer are stored in the usbh0_utr structure member (result).

The transfer results are listed below.

Result	Description
USBH0_CTRL_END	Normal end of control transfer
USBH0_DATA_NONE	Normal end of data transmission
USBH0_DATA_OK	Normal end of data reception
USBH0_DATA_SHT	Normal end of data reception with the data length less than the specified value
USBH0_DATA_OVR	Received data size exceeded
USBH0_DATA_ERR	Non-response condition or overrun/underrun error detected
USBH0_DATA_STALL	STALL response or MaxPacketSize error detected
USBH0_DATA_STOP	Forced end of data transfer

6.4 Retrying Transfer

When a no-response condition occurs on a pipe, the USB-BASIC-F/W performs communication retries. If the no-response condition is detected, USBH0_DATA_ERR is returned to the HDCD.

6.5 Notes on Reception

When a short packet is received, the expected remaining receive data length is stored in tranlen of the usbh0_utr structure and transfer ends. When the received data is larger than the buffer size, data is read from the FIFO buffer up to the buffer size and transfer ends.

6.6 Control Transfer

When the USB-BASIC-F/W firmware is used to transmit requests not listed in section 4.5.2, USB Standard Requests (vendor or class requests) to a device, the following must be set in the usbh0_utr structure member before R_USBH0_HstdTransferStart() is called.

- keyword: Pipe number (USB_PIPE0)
- tranadr: Data buffer (data stage)
- tranlen: Transfer size
- setup: Setup data
- complete: Callback function

The callback function is set to “complete” on completion of data transfer and conveys the result of transfer (see section 6.3, Notification of Transfer Results).

The following is an example of transmission of a vendor request.

<Example of vendor request transmission>

```
uint8_t                 usb_gsmp_VendorRequestData[16];
st_usbh0_setup_t        usb_gsmp_VendorRequest;
usbh0_utr               usb_gsmp_ControlUtr;
usbh0_utr_cb_t          usb_gsmp_VendorRequestCb;

usbh0_er_t   usb_hsmp_VendorRequestProcess(void)
{
   usbh0_er_t err;

   /* Set setup data */
    usb_gsmp_VendorRequest.type         = ((bRequest << 8) | bmRequestType);
    usb_gsmp_VendorRequest.value        = wValue;
    usb_gsmp_VendorRequest.index        = wIndex;
    usb_gsmp_VendorRequest.length       = wLength;
    usb_gsmp_VendorRequest.devaddr      = devaddr;

   /* Set transfer pipe (default pipe) */
   usb_gsmp_ControlUtr.keyword   = USBH0_PIPE0;
   /* Set transmit data */
   usb_gsmp_ControlUtr.tranadr   = usb_gsmp_VendorRequestData;
   /* Set transfer size */
   usb_gsmp_ControlUtr.tranlen   = usb_gsmp_VendorRequest.length;
   /* Set setup command */
   usb_gsmp_ControlUtr.setup     = &usb_gsmp_VendorRequest;
   /* Set callback function */
   usb_gsmp_ControlUtr.complete  = &usb_gsmp_VendorRequestCb;

   /* Data transmission request */
   err = R_USBH0_HstdTransferStart(&usb_gsmp_ControlUtr);

   return     err;
}
```

6.7 Data Transfer

When the USB-BASIC-F/W firmware is used to transfer data, following must be set in the usbh0_utr structure member before R_USBH0_HstdTransferStart() is called.

- keyword : Pipe number
- tranadr : Data buffer
- tranlen : Transfer size
- complete : Callback function

The callback function is set to “complete” on completion of data transfer and conveys the result of transfer (see section 6.3, Notification of Transfer Results).

The following is an example of bulk-in transfer.

<Example of bulk-in transfer>

```
uint8_t             usb_gsmp_VendorBulkInData[512];
usbh0_utr           usb_gsmp_DataUtr;
usbh0_utr_cb_t      usb_gsmp_VendorBulkInCb;

usbh0_er_t   usb_hsmp_VendorBulkInProcess(uint16_t devaddr)
{
   usbh0_er_t err;

   /* Set transfer pipe */
   usb_gsmp_DataUtr.keyword      = R_usb_htsd_GetPipeID(devaddr, USBH0_EP_BULK,
USBH0_EP_IN, 0);
   /* Set transmit data */
   usb_gsmp_DataUtr.tranadr      = usb_gsmp_VendorBulkInData;
   /* Set transfer size */
   usb_gsmp_DataUtr.tranlen      = 512;
   /* Set callback function */
   usb_gsmp_DataUtr.complete     = &usb_gsmp_VendorBulkInCb;

   /* Data transmission request */
   err = R_USBH0_HstdTransferStart(&usb_gsmp_DataUtr);

   return     err;
}
```

7. Scheduler

7.1 Overview

- The scheduler executes task scheduling according to task priority.
- The features of the scheduler are as follows.
- Manages requests by the tasks and hardware according to the priority of the tasks.
- When multiple task requests with the same priority are generated, a FIFO queue determines their order of execution.
- Callback functions are used for responses to tasks indicating the end of a request, so the user need only install appropriate class drivers for the system and there is no need to modify the scheduler itself.

7.2 Scheduler Macro

Table 7-1 lists the scheduler macros available for use by the user.

Table 7-1 Scheduler Macros

Scheduler Macro	Registration Function	port	Description
USBH0_SND_MSG USBH1_SND_MSG	usbh0_hstd_snd_msg usbh1_hstd_snd_msg	0 1	Specifies a message box to transmit a message.
USBH0_WAI_MSG USBH1_WAI_MSG	usbh0_hstd_wai_msg usbh1_hstd_wai_msg	0 1	Runs USBH0_SND_MSG after running the scheduler the specified number of times.
USBH0_RCV_MSG USBH1_RCV_MSG	usbh0_hstd_rec_msg usbh1_hstd_rec_msg	0 1	Checks if the specified mailbox has a received message.
USBH0_PGET_BLK USBH1_PGET_BLK	usbh0_hstd_pget_blk usbh1_hstd_pget_blk	0 1	Secures an area for storing a message.
USBH0_REL_BLK USBH1_REL_BLK	usbh0_hstd_rel_blk usbh1_hstd_rel_blk	0 1	Release an area for storing a message.

7.3 Scheduler API Functions

Table 7-2 lists the API functions for the scheduler.

Table 7-2 Scheduler API Functions

Function Name	port	Description
usbh0_hstd_snd_msg() usbh1_hstd_snd_msg()	0 1	Sends requests for processing to the priority table.
usbh0_hstd_rec_msg() usbh1_hstd_rec_msg()	0 1	Checks if the priority table contains a request for processing.
usbh0_hstd_pget_blk() usbh1_hstd_pget_blk()	0 1	Secures an area for storing a request for processing.
usbh0_hstd_rel_blk() usbh1_hstd_rel_blk()	0 1	Releases an area for storing a request for processing.
R_USBH0_CstdScheduler() R_USBH1_CstdScheduler()	0 1	Manages requests for processing by the tasks.
R_USBH0_CstdSetTaskPri() R_USBH1_CstdSetTaskPri()	0 1	Sets task priority.
R_USBH0_CstdCheckSchedule() R_USBH1_CstdCheckSchedule()	0 1	Checks if the registered task contains a request for processing.

7.3.2 usbh0_hstd_snd_msg

Sending requests for processing

Format

usbh0_er_t	usbh0_hstd_snd_msg(uint8_t id, usbh0_msg_t *mess)

Argument

id	Task ID to send messages
*mess	Message for transmission

Return value

USBH0_OK	Normal end
USBH0_ERROR	Task ID is not set
	Task priority is not set
	Priority table is full (a request cannot be sent to the priority table)

Description

This function transmits requests for processing to the priority table. This function is defined in the USBH0_SND_MSG macro.

Note

1. Call the scheduler macro (USBH0_SND_MSG) in which this function is registered from the user application program other than the interrupt function or the class driver.
2. Specify the address where the memory buffer starts for the message for transmission.
3. The sample program specifies the address where the memory block acquired by the USBH0_PGET_BLK macro starts as that of the message for transmission.

Example

```
void   usb_smp_task()
{
  usbh0_utr         *p_blf;
  usbh0_er_t        err;
    :
  /* Secure a message storage area */
  err = USBH0_PGET_BLK(USB_SMP_MPL, &p_blf);
  if(err == USBH0_OK)
  {
    /* Set up messages */
    p_blf->msginfo      = USB_SMP_REQ;
    p_blf->keyword      = keyword;
    p_blf->complete     = complete;
  }
  /* Send messages */
  err = USBH0_SND_MSG(USB_SMP_MBX, (usbh0_msg_t*)p_blf);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.3.4 usbh0_hstd_rec_msg

Checking if the priority table contains a request for processing

Format

usbh0_er_t	usbh0_hstd_rec_msg(uint8_t id, usbh0_msg_t** mess)

Argument

id	Task ID for reception of messages
*mess	Received mesage

Return value

USBH0_OK	Normal end
USBH0_ERROR	Task ID is not set

Description

This function checks if the priority table contains a request for processing corresponding to the task ID. This function is defined in USBH0_RCV_MSG.

Note

1. Call the scheduler macro in which this function is registered from the user application program or class driver.
2. Do not call this function when the returned value of the R_USBH0_CstdCheckSchedule () function is USBH0_FLGCLR.

Example

```
void   usb_smp_task()
{
  usbh0_utr         *mess;
  usbh0_er_t        err;
    :
  /* Check the message */
  err = USBH0_RCV_MSG(USB_SMP_MBX, (usbh0_msg_t**)&mess);
  if(err == USBH0_OK)
  {
    /* Processing for the message */
  }
    :
}
```

7.3.6 usbh0_hstd_pget_blk

Securing an area for storing requests for processing

Format

usbh0_er_t	usbh0_hstd_pget_blk(uint8_t id, usbh0_utr **blk)

Argument

id	Task ID for securing an area
**blk	Pointer to the area to be secured

Return value

USBH0_OK	The area is secured
USBH0_ERROR	Task ID is not set
	The area is not secured

Description

This function secures an area for storing messages. This API secures 40 bytes per block. This function is registered in USBH0_PGET_BLK.

Note

1. Call the scheduler macro in which this function is registered from the user application program or class driver.

Example

```
void   usb_smp_task()
{
  usbh0_utr         *p_blf;
  usbh0_er_t        err;
    :
  /*Secure a message storage area */
  err = USBH0_PGET_BLK(USB_SMP_MPL, &p_blf);
  if(err == USBH0_OK)
  {
    /* Set up messages */
    p_blf->msginfo      = USB_SMP_REQ;
    p_blf->keyword      = keyword;
    p_blf->complete     = complete;
  }
  /* Send messages */
  err = USBH0_SND_MSG(USB_SMP_MBX, (usbh0_msg_t*)p_blf);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.3.8 usbh0_hstd_rel_blk

Releasing an area for storing requests for processing

Format

usbh0_er_t	usbh0_hstd_rel_blk(uint8_t id, usbh0_utr* blk)

Argument

id	Task ID for releasing an area
*blk	Pointer to the area to be released

Return value

USBH0_OK	The area is released
USBH0_ERROR	Task ID is not set
	The area is not released

Description

This function releases an area for storing messages. The function is registered in USBH0_REL_BLK.

Note

1. Call the scheduler macro in which this function is registered from the user application program or class driver.

Example

```
void   usb_smp_task()
{
  usbh0_er_t        err;
  usbh0_utr         *mess;
    :
  /* Release the message storage area */
  err = USBH0_REL_BLK(USB_SMP_MPL, mess);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.3.10 R_USBH0_CstdScheduler

Managing requests for processing

Format

void	R_USBH0_CstdScheduler(void)

Argument

—	—

Return value

—	—

Description

This function manages requests for processing from the individual tasks

Note

1. Call this function in the loop that executes scheduler processing.

Example

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    /* Task processing */
       :
  }
  /* Start scheduler */
  R_USBH0_CstdScheduler();
       :
}
```

7.3.12 R_USBH0_CstdSetTaskPri

Setting the priority of tasks

Format

void	R_USBH0_CstdSetTaskPri(uint8_t tasknum, uint8_t pri)

Argument

tasknum	Task ID
pri	Task priority

Return value

—	—

Description

This function sets the priority of tasks. The priority (pri) specified by the 2nd argument is set in the task specified by the 1st argument (tasknum).

Note

1. Call this function from the user application program at the initial setup. Messages can be transferred by setting the task ID priorities.

Example

```
void usb_smp_driver_start(void)
{
  /* Set the priority of the sample task to 4 */
  R_USBH0_CstdSetTaskPri(USB_SMP_TSK, USB_PRI_4);
}
```

7.3.14 R_USBH0_CstdCheckSchedule

Checking whether or not processing is scheduled

Format

uint8_t	R_USBH0_CstdCheckSchedule(void)

Argument

—	—

Return value

USBH0_FLGSET	Request for processing
USBH0_FLGCLR	No request for processing

Description

This function checks if the registered task contains a received message.

Note

1. Call this function in the loop that executes scheduler processing.

Example

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    /* Task processing */
       :
  }
  /* Start scheduler */
  R_USBH0_CstdScheduler();
       :
}
```

8. Reference Documents

User's Manual: Hardware

RZ/A2M Group User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

RTK7921053C00000BE (RZ/A2M CPU board) User's Manual
The latest version can be downloaded from the Renesas Electronics website.

RTK79210XXB00000BE (RZ/A2M SUB board) User's Manual
The latest version can be downloaded from the Renesas Electronics website.

ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
The latest version can be downloaded from the ARM website.

ARM Cortex™-A9 Technical Reference Manual Revision: r4p1
The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification - Architecture version 2.0
The latest version can be downloaded from the ARM website.

ARM CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3
The latest version can be downloaded from the ARM website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools

Integrated development environment e2studio User’s Manual can be downloaded from the Renesas Electronics website.
The latest version can be downloaded from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.15.19	-	First edition issued
1.10	May.17.19	6	Table 2.1 Operation Confirmation Conditions(1/2) Remove compiler option "-mthumb-interwork"
1.20	Dec.17.19	6	Table 2.1 Operation Confirmation Conditions(1/2) Change compiler option "-g3" to "-None" Support both FreeRTOS / OSLess

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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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 claims 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, drawings, charts, programs, algorithms, and 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 reverse engineer 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, copying or reverse engineering.
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; 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.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; 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 any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using 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 dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of 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. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a 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. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using 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 comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.