Renesas USB MCU

USB Basic Firmware mini

R01AN0326EJ0201
Rev.2.01
Mar 26, 2013

Introduction

This document is an application note describing the USB Basic Firmware mini, a sample program for USB interface control using the Renesas USB MCU.

Target Device

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

Contents

1. Overview

This application note describes the USB Basic Firmware mini using the Renesas USB MCU.

This document is intended to be used together with the device's data sheet of Chapter1.2

1.1 Functions and Features

The USB Basic Firmware mini conforms to the Full-Speed and Low-Speed of Universal Serial Bus Specification (USB from now on and description). It and enables communication with a USB vendor host or USB vendor peripheral device.

1.2 Related Documents

1. Universal Serial Bus Revision 2.0 specification
2. Battery Charging Specification Revision 1.2
   [http://www.usb.org/developers/docs/]
3. Renesas USB MCU User's Manual: Hardware
   Available from the Renesas Electronics Website

- Renesas Electronics Website
  [http://renesas.com/]
- USB Devices Page
  [http://renesas.com/usb/]

1.3 List of Terms

Terms and abbreviations used in this document are listed below.

API	:	Application Program Interface
APL	:	Application program
cstd	:	Prefix for peripheral & host common function of USB-BASIC-F/W
CubeSuite+	:	Renesas integration development environment (for RL78)
CDP	:	Charging Downstream Port
DCP	:	Dedicated Charging Port
HBC	:	Host Battery Charging control
Data Transfer	:	Generic name of Bulk transfer and Interrupt transfer (When the host mode is selected, the Control transfer is contained.)
EVA	:	Evaluation board
EWRL78		IAR Embedded Workbench for RL78 IAR integration development environment (for RL78)
E2Studio	:	Eclipse embedded studio (However not correspondence)
HCD	:	Host control driver of USB-BASIC-F/W
HDCD	:	Host device class driver (device driver and USB class driver)
HEW	:	High-performance Embedded Workshop
HM	:	Hardware Manual
hstd	:	Prefix for host function of USB-BASIC-F/W
H/W	:	Renesas USB device
MGR	:	Sequencer of HCD to manage state of the peripheral device
PBC	:	Peripheral Battery Charging control
PCD	:	Peripheral control driver of USB-BASIC-F/W
PDCD	:	Peripheral device class driver (device driver and USB class driver)
PP	:	Pre-processed definition
pstd	:	Prefix for peripheral function of USB-BASIC-F/W
RSK	:	Renesas Starter Kit
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function

SDP	:	Standard Downstream Port
Task	:	Processing unit
UPL		User Programming Layer (Upper layer of USB-BASIC-F/W:HDCD, PDCD, APL or etc)
USB	:	Universal Serial Bus
USB-BASIC-F/W	:	USB Basic Firmware mini (Peripheral & Host USB basic firmware(USB low level) for Renesas USB MCU)

1.4 How to Read This Document

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

To get acquainted with the source code, read Chapter 4.3 and note which MCU-specific files you need select at directory "*\Project***devicename***\HwResourceForUSB*". Observe which files belong to the application level.

Chapter 5 and Chapter 6 of this document are only for the peripheral mode. Chapter 7 and Chapter 8 of this document are only for the host mode. Chapter 5 explains how the default peripheral vendor application works. Chapter 7 explains how the default host vendor application works. You will change this to create your own solution.

Understand how all code modules are divided into tasks, and that these tasks pass messages to one another. This is so that functions (tasks) can execute in the order determined by a scheduler and not strictly in a predetermined order. This way more important tasks can have priority. Further, tasks are intended to be non-blocking by using a documented callback mechanism. The task mechanism is described in Chapter 9.1. All USB-BASIC-F/W tasks are listed in Chapter 4.4.

2. How to Register Class Driver

The class driver which the user created functions as the USB class driver by registering with the USB-BASIC-F/W.

2.1 How to register Peripheral Class Driver

Please consult function *usb_psmpl_driver_registration()* in *r_usb_vendor_papl.c* and register the class driver into a USB-BASIC-F/W. For details, please refer to the Chapter 6.

The following describes how to register user-created class drivers and applications in the USB Basic Firmware mini.

```
USB_STATIC void usb_psmpl_driver_registration(void)
{
  usb_pcdreg_t driver;
  /* Driver registration */
  driver.pipetbl   = g_usb_psmpl_EpTbl1;             /* Pipe define table */
  driver.devicetbl = g_usb_psmpl_DeviceDescriptor;
  driver.configtbl = g_usb_psmpl_Configuration;
  driver.stringtbl = g_usb_psmpl_StringPtr;
  driver.statediagram = &usb_psmpl_device_state;     /* Change device state */
  driver.ctrltrans = &usb_psmpl_control_transfer;    /* Control transfer */
  R_usb_pstd_DriverRegistration(&driver);
}
```

2.2 How to register Host Class Driver

Please consult function *usb_hsmpl_driver_registration()* in *r_usb_vendor_hapl.c* and register the class driver into a USB-BASIC-F/W. For details, please refer to the Chapter 8.

The following describes how to register user-created class drivers and applications in the USB Basic Firmware mini.

```
USB_STATIC void usb_hsmpl_driver_registration(void)
{
  usb_hcdreg_t driver;
  /* Driver registration */
  driver.ifclass      = USB_IFCLS_VEN;               /* Device class */
  driver.classcheck   = &usb_hsmpl_class_check;      /* Operation judgment */
  driver.statediagram = &usb_hsmpl_device_state;     /* Change device state */
  R_usb_hstd_DriverRegistration(&driver);
}
```

3. Development Goals

USB-BASIC-F/W was developed with the following goals in mind.

- To simplify the development of USB communication programs by customers using the Renesas USB MCU.
- To provide source code examples for hardware control of USB.

3.1 Features of USB-BASIC-F/W

The main features of USB-BASIC-F/W as sample firmware for the H/W control with built-in device are as follows.

3.1.1 Overall

- Capable of FullSpeed/LowSpeed of the USB2.0 specification.
- Can control R8C/USB and RL78/USB by common source code. (Refer to Table 3-1 for the MCU difference.)
- Can operate in either USB host mode or USB function mode.
- The API function of the H/W control (for devices connect/disconnect processing, suspend/resume processing, and remote wakeup processing) is provided.
- The API function of the Data transfer (for control transfer, bulk transfer and interrupt transfer processing) is provided.
- Two or more data transfers are possible according to exclusive pipe use by the same pipe, because UPL (User Programming Layer) manages the data toggle of the end point.
- The callback function to notify UPL the result of H/W control, the result of data transfer, and the USB state transition can be registered.
- The sample application and the vendor class driver that shows the usage example of USB-BASIC-F/W are provided.
  (1) The control transfer (Enumeration processing)
  (2) The bulk transfer and interrupt transfer
  (3) A method of describing the class request (control transfer)

3.1.2 Host mode

- In host mode, enumeration with low-speed/full-speed device
  (Corresponding to connect the Low-Speed device is only RL78/USB.)
- A sample program for control transfer (Enumeration processing) is provided.
- A common data transfer API function (for the control transfer, the bulk transfer, and the interrupt transfer) is provided.
- The API function for suspend processing and resume processing is offered.
- A sample program for CDP operation or DCP operation is provided (Only RL78/USB).

3.1.3 Peripheral (function) mode

- In peripheral mode, enumeration is possible as low-speed/full-speed device with USB Host of USB1.1/2.0/3.0.
  (Low-Speed device is only possible with RL78/USB.)
- Operation can be confirmed by using *USBCommandVerifier.exe*.
  (USBCV is available for download from http://www.usb.org/developers/tools/.)
  Normally, a hub must be HS in order for USB-CV to work. Connect HS hub between PC and device.
- A sample program for control transfer (enumeration processing) is provided.
- An API for the FIFO buffer access when the control transfer is provided.
- A common data transfer API function for bulk transfer and the interrupt transfer is provided.
- An API function for remote wakeup is provided.
- A sample program for CDP operation is provided (Only RL78/USB).

3.1.4 Functionality provided by user

The following functions must be provided by the customer to match the system under development.

- Over current detection processing and descriptor analysis (Host mode).
- Device class driver example currently exists for HID, MSC, CDC, LibUSB, etc.
- The pipe information table.
- The descriptor table (peripheral mode)

3.2 Scheduler Function and Tasks

The scheduler function manages requests generated by tasks, according to the task ID, and requests occurring due to H/W interrupt. USB-BASIC-F/W notifies a task about the end of request via a callback function. The scheduler function does not have to change when adding or changing the UPL. Please refer to Chapter 9.1 for details of the scheduler function.

3.3 Function difference by MCU

Table 3-1 shows functional difference by MCU.

Table 3-1 USB functional difference list by MCU

Function	R8C/USB	RL78/USB
MCU type	R8C/34U, R8C/3MU, R8C/34K, R8C/3MK	RL78/G1C
Peripheral mode Transmission rate possible	1Port Full-Speed device	1Port*1 Full-Speed device Low-Speed device
Host mode Number of port and transmission rate that can be connected	R8C/34K, R8C/3MK are 1Port Host R8C/34U, R8C/3MU peripheral only Full-Speed device	2PortHost*2 For Full-Speed device For Lowl-Speed device
Control transfer	PIPE0	PIPE0
Bulk transfer	PIPE4, PIPE5	PIPE4, PIPE5
Interrupt transfer	PIPE6, PIPE7	PIPE6, PIPE7
To connect HUB device when host mode	Not connected	Not Connected
PIPE function	Set at only endpoint number	Set at only endpoint number
Battery Charging	Not available	Available

Note)

*1: The user can customize whether to operate the FullSpeed peripheral device or the LowSpeed peripheral device in the USB-BASIC-F/W and UPL. Please refer to Chapter 5.6 for details.

*2: When the target board is the RL78-RSK, the host mode operation is a possible port only on the *USB-PORT1* side. However, it is necessary to build the USB-BASIC-F/W by 2PORTHOST for the USB-PORT1 side is the host mode operation. Please refer to Chapter 7.5 for details.

4. Software Configuration

4.1 Module Configuration

Software that composes USB-BASIC-F/W has a "*Task*" structure. The task hierarchy of the USB-BASIC-F/W is shown in Figure 4.1 and the software functional overview is shown in Table 4-1. These tasks communicate via the scheduler using a messaging system.

The USB-BASIC-F/W is composed of PCD (peripheral control driver), HCD (host control driver), and MGR (processes the USB state management and host sequence). The USB class driver (HDCD/PDCD), the host device driver (HDD) and an application (APL) are not a part of USB-BASIC-F/W.

The PCD operates the H/W control and the data transfer from the demand of UPL. Also notify to the task when the H/W control ends, the result of data transfer, and of requests of interrupt handler (status change etc).

The HCD operates the H/W control and the data transfer from the demand of MGR task. Also operates the data transfer from the demand of UPL. Notify to the MGR task when the H/W control ends and of requests of interrupt handler (status change etc). And notify to the MGR task and UPL the result of data transfer.

MGR manages the USB state of the connected device and processes sequences such as enumeration. Moreover, the USB state of the connected device changes according to the demand of UPL by API functions. To do this, MGR sends requests to HCD to achieve sequence processing (H/W control and data transfer) necessary for USB state transition. The result of the USB state transition is notified to UPL via callbacks.

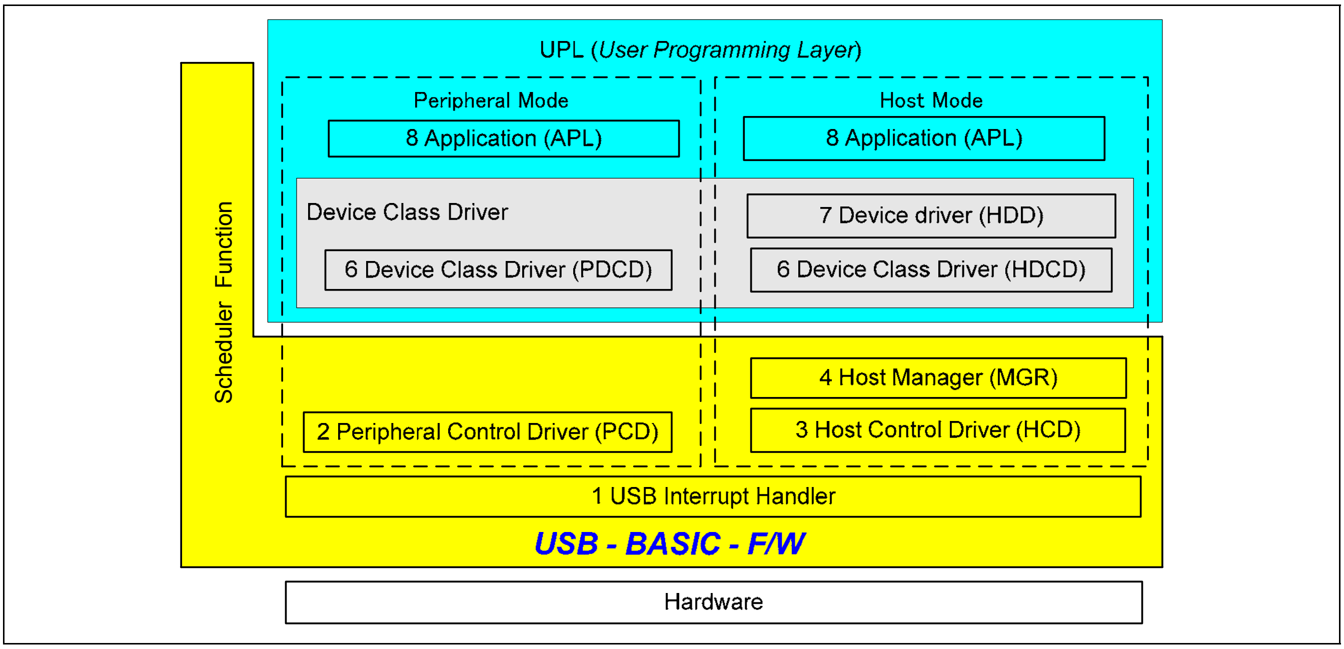

Figure 4.1 Task Configuration of USB-BASIC-F/W

Table 4-1 Software function overview

No	Module Name	Description	Notes
1	USB Interrupt Handler	USB interrupt handler (USB packet transmit/receive end and special signal detection)	
2	Peripheral Control Driver (PCD)	Hardware control in peripheral mode Peripheral transaction management	
3	Host Control Driver (HCD)	Hardware control in host mode Host transaction management	
4	Host Manager (MGR)	Device state management Enumeration	
5	Device Class Driver (PDCD/HDCD)	Provided by the customer as appropriate for the system. (Rensas class driver examples are available for download)	
6	Host Device Driver (HDD)	Provided by the customer as appropriate for the system. (Rensas class driver examples are available for download)	
7	Application(APL)	Provided by the customer as appropriate for the system. (Rensas APL examples are available for download)	

4.2 Overview of Application Program Functions

The main functions of the application are to.

1. Enumerates to the connected USB device.
2. The data is received from the connected USB device by the bulk transfer and the interrupt transfer.
3. The data is transmitted to the connected USB device by the bulk transfer and the interrupt transfer.
4. The device state of the connected USB device is changed when user presses SW1 to SW3 on the RL78-RSK.

When the peripheral device is the LowSpeed device, only the interrupt transfer is possible to the data communication.

Switch input operation is described in Table 4-2 and Table 4-3.

Table 4-2 User switch input operation of the host mode

Switch Function	Description	Switch Number
SUSPEND	The connected peripheral device is suspended	SW1
RESUME	The connected peripheral device is resumed	SW2
PORTCONTROL	VBUS output is disable	SW3

Table 4-3 User switch input operation of the peripheral mode

Switch Function	Description	Switch Number
REMOTEWAKEUP	The connected host device is wakeuped	SW1
PORT OFF	Pull-up release of D+ or D- line	SW2
PORT ON	Pull-up sets of D+ or D- line	SW3

4.3 Structure of Files and folders

4.3.1 Folder Structure

The folder composition of the files that USB-BASIC-F/W provides in is shown below. The offered file includes vendor class driver for the data transfer, application and hardware resource sample code.

The project file of the integration environment, the source code that controls the MCU and the evaluation board are stored under the Project folder.

```
• USBFWmini
  +WorkSpace [Hew/CubeSuite+/EWRL78 WorkSpace]
  |   (USB-BASIC-F/W Code) [Common USB code that is used by all USB firmware]
  |   +―――USBSTDFW
  |   |          +――― inc                     Common header file of USB driver
  |   |          +――― src                     USB driver
  |   (Sample Code) [Class driver and sample application]
  |   +―――SmplMain
  |   |          +――― APL                     Sample application
  |   +―――VENDOR  [Vendor Class driver]      See Table 4-4
  |   |          +――― inc                     Common header file of vendor class driver
  |   |          +――― src                     Vendor class driver
  +Project (Hardware Setting) [Hardware access layer; to initialize the MCU]
      +―――R8C
      |          +――― HwResourceForUSB        Hardware resource for R8C/USB
      |          +――― Object                  Build result
      +―――RL78
                 +――― HwResourceForUSB_G1C     Hardware resource for RL78/USB
                 +――― HwResourceForUSB_G1C_EVA Hardware resource for RL78 evaluation board
                 +――― HwResourceForUSB_G1C_RSK Hardware resource for RL78-RSK
                 +――― Object                  Build result
```

Please change the H/W resource folder name of the board used from "*HwResourceForUSB_G1C _ **board name***" to "*HwResourceForUSB_G1C*" when MCU is RL78.

4.3.2 List of files

The files provided in USB-BASIC-F/W are listed below.

Table 4-4 List of source file

Folder	File Name	Description	Notes
USBSTDFW\src	r_usb_cstdapi.c	USB library API functions	
USBSTDFW\src	r_usb_cstdfunction.c	USB library functions	
USBSTDFW\src	r_usb_h1port.c	The 1 port host functions	
USBSTDFW\src	r_usb_h2port.c	The 2 port host functions	
USBSTDFW\src	r_usb_hbc.c	USB HBC control functions	
USBSTDFW\src	r_usb_hdriver.c	USB Host Control Driver	
USBSTDFW\src	r_usb_hdriverapi.c	HCD API functions	
USBSTDFW\src	r_usb_hp0function.c	Potr0 control functions	
USBSTDFW\src	r_usb_hp1function.c	Potr1 control functions	
USBSTDFW\src	r_usb_pbc.c	USB PBC control functions	
USBSTDFW\src	r_usb_pdriver.c	USB Peripheral Control Driver	
USBSTDFW\src	r_usb_pdriverapi.c	PCD API functions	
USBSTDFW\src	r_usb_hport.h	Prototype declaration of USB host functions	
USBSTDFW\src	r_usb_iodefine.h	Macro definition of USB register access	
USBSTDFW\inc	r_usb_api.h	Prototype declaration of USB API functions	
USBSTDFW\inc	r_usb_cdefusbip.h	Macro definition of USB-BASIC-F/W	
USBSTDFW\inc	r_usb_ckernelid.h	Macro definition of scheduler functions	
USBSTDFW\inc	r_usb_ctypedef.h	Type definition of USB-BASIC-F/W	
USBSTDFW\inc	r_usb_usrconfig.h	Macro definition of user configuration	
SmplMain	main.c	Main process	
SmplMain\APL	r_usb_vendor_descriptor.c	Descriptor and Endpoint information	
SmplMain\APL	r_usb_vendor_hapl.c	Host sample application program	
SmplMain\APL	r_usb_vendor_papl.c	Peripheral sample application program	
SmplMain\APL	r_usb_vendor_apl.h	Macro define of application	
VENDOR\src	r_usb_vendor_hapi.c	Sample HDCD API	
VENDOR\src	r_usb_vendor_hdriver.c	Sample HDCD	
VENDOR\src	r_usb_vendor_papi.c	Sample PDCD API	
VENDOR\src	r_usb_vendor_pdriver.c	Sample PDCD	
VENDOR\inc	r_usb_vendor_api.h	Prototype declaration of Vendor class driver	
Project\R8C\HwResourceForUSB\src	ncrt0.a30 adc_driver_r8c.c lcddriver_r8c.c r8cusbmcu.c iodefine_r8c.h nc_define.inc sect30.inc	Startup program AD converter driver LCD driver MCU control processing IO define header Macro Symbol definition Section define	
\inc	hw_resource.h	Prototype declaration of special function driver	
Project\RL78\HwResourceForUSB\src	cstartn.asm rom.asm adcdriver_rl78.c keydriver_rl78.c lcddriver_rl78.c leddriver_rl78.c rl78usbmcu.c	Startup program Section define AD converter driver KEY driver LCD driver LED driver MCU control processing	
\inc	hw_resource.h	Prototype declaration of special function driver	

4.4 System Resources

4.4.1 System Resource Definitions

Table 4-5 and Table 4-6 list the Task ID and the task priority definitions used to register USB-BASIC-F/W in the scheduler. These are defined in the *r_usb_cKernelId.h* header file.

Table 4-5 List of Scheduler Registration IDs when host operates

Scheduler registration task	Description	Notes
Task ID: USB_HVEN_TSK	**HDCD** (R_usb_hvndr_Task) Task priority: 2	
Task ID: USB_HSMP_TSK	**APL** (usb_hsmpl_apl_task) Task priority: 3	
Task ID: USB_HCD_TSK	**HCD** (R_usb_hstd_HcdTask) Task priority: 0	
Task ID: USB_MGR_TSK	**MGR** (R_usb_hstd_MgrTask) Task priority: 1	
Mailbox ID / Default receive task	**Message description**	**Notes**
USB_HVEN_MBX / USB_HVEN_TSK	APL -> HDCD mailbox ID	
USB_HSMP_MBX / USB_HSMP_TSK	HDCD -> APL mailbox ID	
USB_HCD_MBX / USB_HCD_TSK	HCD task mailbox ID	
USB_MGR_MBX / USB_MGR_TSK	MGR task mailbox ID	

Table 4-6 List of Scheduler Registration IDs when peripheral operates

Scheduler registration task	Description	Notes
Task ID: USB_PVEN_TSK	**PDCD** (R_usb_pvndr_Task) Task priority: 3	
Task ID: USB_PSMP_TSK	**APL** (usb_psmpl_apl_task) Task priority: 4	
Task ID: USB_PHCD_TSK	**PCD** (R_usb_pstd_PcdTask) Task priority: 0	
Mailbox ID / Default receive task	**Message description**	**Notes**
USB_PVEN_MBX / USB_PVEN_TSK	APL -> PDCD mailbox ID	
USB_PSMP_MBX / USB_PSMP_TSK	PDCD -> APL mailbox ID	
USB_PCD_MBX / USB_PCD_TSK	PCD task mailbox ID	

4.5 Note

The customer will need to make a variety of customizations, for example corresponding to class request and vender request, difference of system configuration, processing strengthening when abnormality generated, to consider the transmission rate and the program capacity, or making individual settings that affect the user interface.

Note: USB-BASIC-F/W is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to various USB devices.

5. Peripheral Sample Program (UPL)

This chapter assumes and explains the case where RL78 is used as MCU. The LowSpeed device cannot communicate the bulk transfer. Skip the description concerning the bulk transfer when the user system is LowSpeed device. R8C doesn't correspond to the LowSpeed device. Skip the description concerning the LowSpeed when the user system uses R8C as MCU.

The sample application performs the data communication when connected to the USB device.

5.1 Operating Environment

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

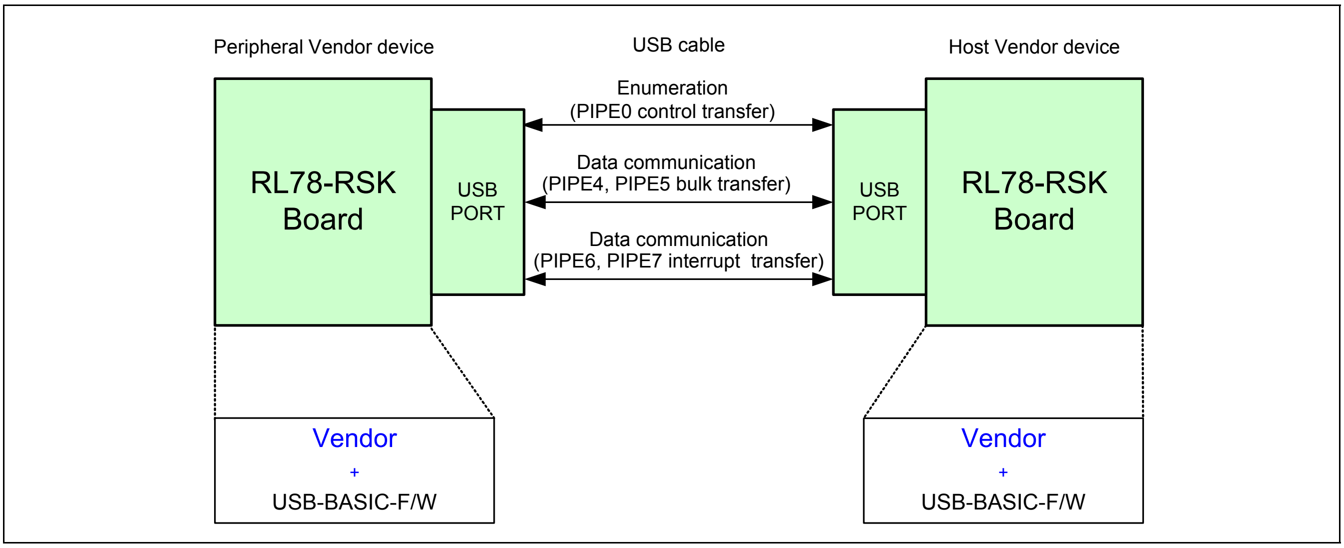

Figure 5.1 Example FullSpeed Operating Environment

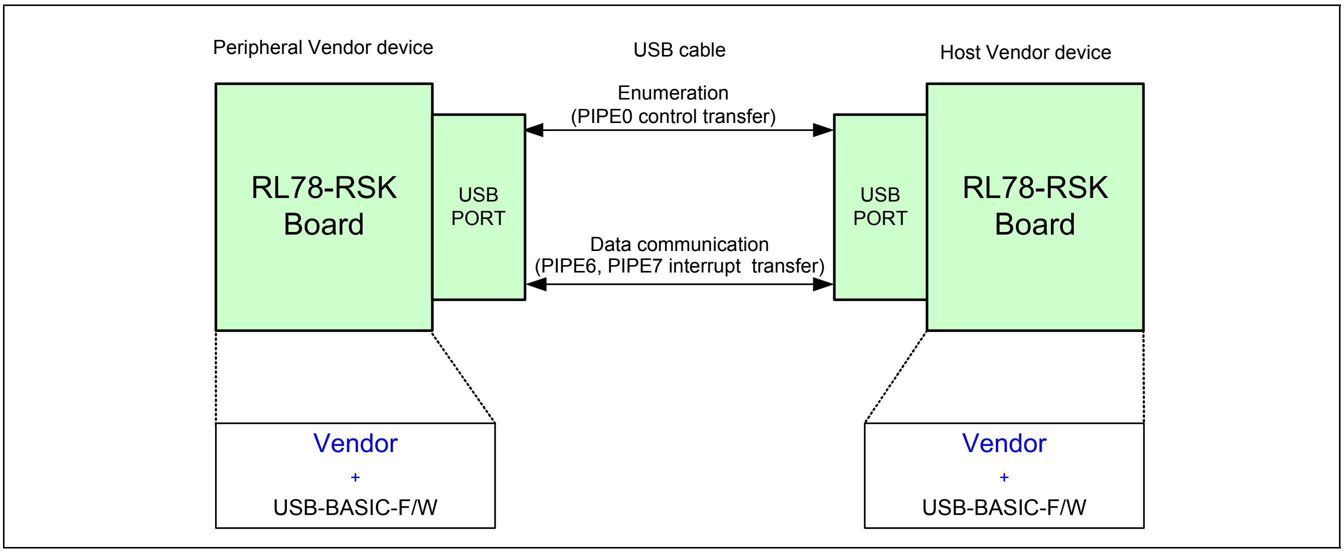

Figure 5.2 Example LowSpeed Operating Environment

5.2 Description of Peripheral Sample Program

The peripheral sample program of the USB-BASIC-F/W is operated with the FullSpeed device or the LowSpeed device that can be selected by the user specification in the *r_usb_usrconfig.h* file. A sample program includes a vendor class driver and sample application for the data transfer. The data communication of bulk transfer uses the pipes 4 and 5. And the data communication of interrupt transfer uses the pipes 6 and 7. When creating a customer class driver or an application, refer to the *r_usb_vendor_papl.c* file, *r_usb_vendor_descriptor.c* file, and *r_usb_vendor_pdriver.c* file. The following settings are necessary in order to communicate with a USB host as a USB peripheral device.

1. Select operation with the FullSpeed device or the LowSpeed device.
2. Setting a scheduler (the number of tasks, table size, task ID, and mail box ID, etc.)
3. Calling a task
4. Creating a corresponding descriptor table to a device class driver to be mounted
5. Creating a corresponding pipe information table to a device class driver to be mounted
6. Returning a USB request

5.2.1 Functions

Sample application

The USB state transition that is callback from the USB-BASIC-F/W is notified to the vendor class driver. And control the vendor class driver. At this time, when USB_STS_CONFIGURED is notified from the USB-BASIC-F/W, the global variable is initialized and the data transfer beginning is demanded to the vendor class driver. The data communications are the bulk transfer using PIPE4 and 5, and interrupt transfer using PIPE6 and 7. When the end of data transfer is notified by the vendor class driver, the data transfer is restarted with the pipe that undertakes the notification. When USB_STS_SUSPEND is notified from the USB-BASIC-F/W, the APL executes the STOP instruction. Moreover, the notification of the key input is received by regular processing. The sample of remote wake up (in suspend state), and the port enable or disable are included.

Vendor class driver

The global variable is initialized according to the USB state that is notified from APL. Moreover, when the data transfer is demanded from the application, the data transfer is demanded to USB-BASIC-F/W. The end of data transfer is notified to the application when the end of data transfer is notified from USB-BASIC-F/W. It doesn't correspond to the vendor class request.

Enumeration

When the USB host's connection is detected, USB-BASIC-F/W automatically starts enumeration. An enumeration ends normally, if a vendor class driver is in the USB host. And USB_STS_CONFIGURED is notified to the application by the callback function.

Data communication

When an enumeration normally ends, the data transfer is possible. The application begins the data transfer of the USB state transition callback. It is possible to communicate with the device that operates USB-BASIC-F/W as the USB host mode.

Vendor class request

A vendor class request is not issued. (STALL response)

USB state transition

To operates as follows by the notification of the USB state transition.

USB_STS_DETACH:	Stop the data transfer
USB_STS_DEFAULT:	Initialized data transfer size, Initialized configuration number
USB_STS_ADDRESS:	Initialized configuration number
USB_STS_CONFIGURED:	Initialized data toggle buffer, Start the data transfer
USB_STS_SUSPEND:	Interrupt the data transfer, Execute the STOP instruction
USB_STS_RESUME:	Restart the data transfer

It is possible to return from the state of the suspended by the resume signal. Moreover, it is also possible to demand remote wake up from the application to USB-BASIC-F/W.

USB device framework

Operation can be confirmed using a device framework test with USBCommandVerifier.exe (USBCV) distributed from the USB Implementers Forum (USB-IF). A supported test item is Chapter 9 only.

5.2.2 Operation of Peripheral Sample Program

1. Initialization
   - For HEW

   When performing hardware reset for a device, the *_PowerON_Reset_PC* function in *ncrt0.a30* is called. The reset function initializes the MCU and call the hardware initialization function *usb_cpu_mcu_initialize()* function. When returning from the hardware initialization function, initialize memory areas and calls the *main()* function in *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

   - For CubeSuite+

   When performing hardware reset for a device, the *_@cstart* function of a startup file created using the CubeSuite+ is called. The startup function initializes the MCU, and call the hardware initialization function *hdwinit()* function of the user definition. When returning from the hardware initialization function, initialize memory areas such as *sadd*r area and call the *main()* function in the *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

   - For EWRL78

   When performing hardware reset for a device, the *_@cstart* function in cstartup.s87 is called. The startup function initializes the MCU, and call the hardware initialization function *hdwinit()* function of the user definition. When returning from the hardware initialization function, initialize memory areas such as *sadd*r area and call the *main()* function in the *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

2. main function processing
   The *main()* function initializes the system by the *usb_psmpl_main_init()* function(initialization of target MCU and the board, initialization of the USB module, starts of USB-BASIC-F/W, registration of the UPL driver's, and operation permission of the USB module), and the program is in the static state that wait for a request generation in the main loop.
   Operations of the main loop are as follows:
   (1) Determine a request in a scheduler.
   (2) When processing is requested, start a task.
   (3) Perform static processing.
   (4) Return to (1).

3. Sample application task (*usb_psmpl_apl_task()*)
   When an enumeration normally ends, the sample application initializes global variables and requests the start of data transfer using API function *R_usb_pvndr_TransferStart()* to the vendor class driver. When a transfer end callback is received from the vendor class driver, the data transfer is repeated using API function *R_usb_pvndr_TransferStart()*.When USB_STS_SUSPEND is notified from the USB-BASIC-F/W, the APL executes the STOP instruction by the *usb_cpu_stop_mode()* function.

4. Vendor class driver (*R_usb_psmpl_VendorTask()*)
   When the data transfer demand is notified from the sample application, the vendor class driver (PDCD) demands the data transfer to USB-BASIC-F/W using API function *R_usb_pstd_TransferStart()*. Moreover, the end of the data transfer is notified to the application by the callback function when the forwarding end callback is received from USB-BASIC-F/W.
   When the USB state transition is notified from the sample application, by he or she the vendor class driver initializes the following global variables according to the USB state.

   USB_STS_CONFIGURED
   Keep the configuration number, and initialize the global variable of the DATA-PID table.
   USB_STS_DETACH, USB_STS_ ADDRESS, USB_STS_ DEFAULT
   "0" cleared of configuration number.
   USB_STS_SUSPEND, USB_STS_RESUME
   No processing.

Figure 5.3 shows the outline flow of the UPL.

The USB-BASIC-F/W comprises tasks that implement control functions for USB data transmit and receive operations. When an interrupt occurs, a notification is sent by means of a message to the USB-BASIC-F/W. When the USB-BASIC-F/W receives a message from the USB interrupt handler, it determines the interrupt source and executes the appropriate processing.

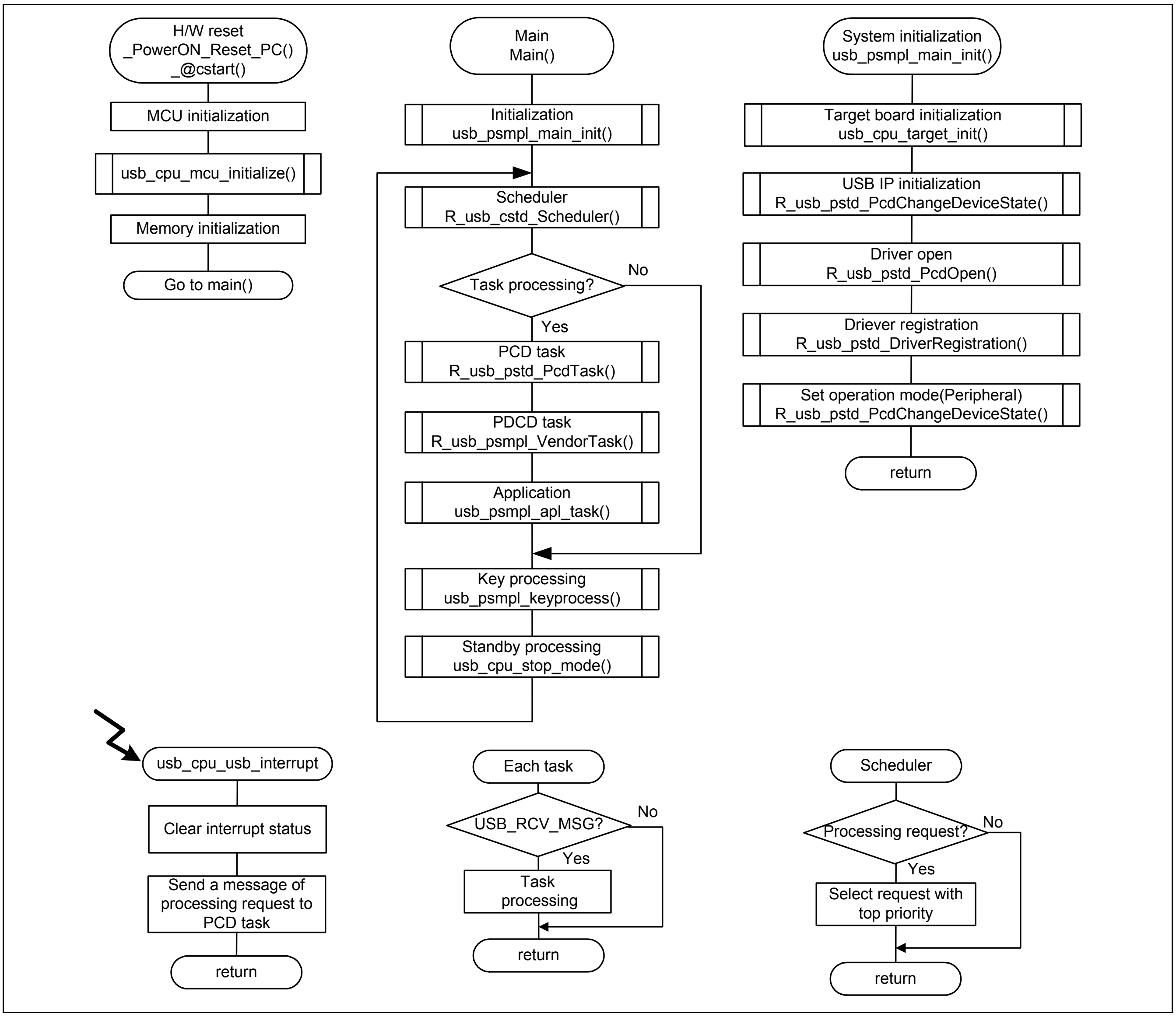

Figure 5.3 Sequence Outline

5.2.3 Setting a Scheduler

Set the maximum value of a task ID, and maximum value of a message stored in the task priority table at *r_usb_cstd_kernelid.h* file.

```
/* Please set with user system */
#define    USB_IDMAX       ((uint8_t)5)    /* Maximum Task ID +1 */
#define    USB_TABLEMAX    ((uint8_t)5)    /* Maximum priority table */
#define    USB_BLKMAX      ((uint8_t)5)    /* Maximum block */
```

5.2.4 Setting a Task ID and Mail Box ID

Set a task ID and mail box ID at *r_usb_cstd_kernelid.h* file..
The task priority level is the same as task ID. (When the task identification number is small, priority is high.)

```
#define    USB_PCD_TSK     USB_TID_0       /* Peripheral Control Driver Task */
#define    USB_PCD_MBX     USB_PCD_TSK     /* Mailbox ID */
#define    USB_PVEN_TSK    USB_TID_3       /* Vendor Class Driver ID */
#define    USB_PVEN_MBX    USB_PVEN_TSK    /* Mailbox ID */
#define    USB_PSMP_TSK    USB_TID_4       /* Peripheral Sample Application Task */
#define    USB_PSMP_MBX    USB_PSMP_TSK    /* Mailbox ID */
```

5.2.5 Task calling

Call a task to be used in main loop (*main()* function).

```
void main(void)
{
  /* Initialized USBIP */
  usb_psmpl_main_init();

  /* Sample main loop  */
     while( 1 )
     {
         if( R_usb_cstd_Scheduler() == USB_FLGSET )
         {
             R_usb_pstd_PcdTask();       /* PCD Task */
             R_usb_psmpl_VendorTask();
             usb_psmpl_apl_task();
         }
         keydata = usb_smpl_KeyRead();
         if (keydata != 0x00)
         {
             usb_psmpl_keyprocess(keydata);
         }
         if ( g_usb_suspend_flag == USB_YES )
         {
             usb_cpu_stop_mode();
         }
     }
```

5.2.6 Starting the UPL

When the USB-BASIC-F/W establishes a structure with a host (SET_CONFIGURATION request received), a device connection is notified to the UPL using the callback function (**g_usb_PcdDriver.statediagram*). Analyze the USB sate of the second argument and perform the suited processing to a system. The sample application notifies the USB state to vendor class driver, initializes the data area, and starts data transfer. The vendor class driver of a sample initializes the data area and starts data communication. The vendor class driver memorizes the configuration number when the USB state transition is notified and initializes.

5.2.7 Returning USB Request

A program example of the control transfer using the API function provided by USB-BASIC-F/W is shown below. The control transfer is shown in the example when a class request is received.

```
void usb_psmp_ControlTransfer(usb_request_t* request, uint16_t ctsq)
{
  g_usb_psmp_Request  = request;
  if ((g_usb_psmp_Request.wRequest & USB_BMREQUESTTYPETYPE) == USB_CLASS)
  {
    switch( ctsq )
    {
        case USB_CS_IDST: usb_psmp_control_trans0(request);break;
        case USB_CS_RDDS: usb_psmp_control_trans1(request);break;
        case USB_CS_WRDS: usb_psmp_control_trans2(request);break;
        case USB_CS_WRND: usb_psmp_control_trans3(request);break;
        case USB_CS_RDSS: usb_psmp_control_trans4(request);break;
        case USB_CS_WRSS: usb_psmp_control_trans5(request);break;
        case USB_CS_SQER:
            R_USB_pstd_ControlEnd((uint16_t)USB_DATA_ERR);   break;
        default:
            R_USB_pstd_ControlEnd((uint16_t)USB_DATA_ERR);   break;
    }
  }
  else
  {
    R_USB_pstd_SetStallPipe0();
  }
}
```

1. Data stage processing

   Transfer the data to a USB host using the API function of *R_usb_pstd_ControlRead()*/*R_usb_pstd_ControlWrite()* for a supported request. Call the API function of *R_usb_pstd_SetStallPipe0()* and return STALL to a USB host for an unsupported request.

2. Status stage processing

   When the data stage properly ends, specify USB_CTRL_END to the argument in status stage and call the API function of *R_usb_pstd_ControlEnd()*. When the data stage does not properly end, specify SB_DATA_ERR to the argument in the data stage and call the API function of *R_usb_pstd_ControlEnd()*.

3. Note

   USB-BASIC-F/W accesses the user buffer up to the data size specified with API function *R_usb_pstd_ControlRead()* / *R_usbh_pstd_ControlWrite()*. Therefore, to make sure that the capacity of the user buffer exceeds transmit / receive data size of the control transfer data stage.

5.2.8 Application Outline

The USB-BASIC-F/W starts the data transfer after configuration in the procedure shown below.
Identify the USB state using callback function *usb_psmpl_device_state()*, and requests to vendor class driver the data transfer

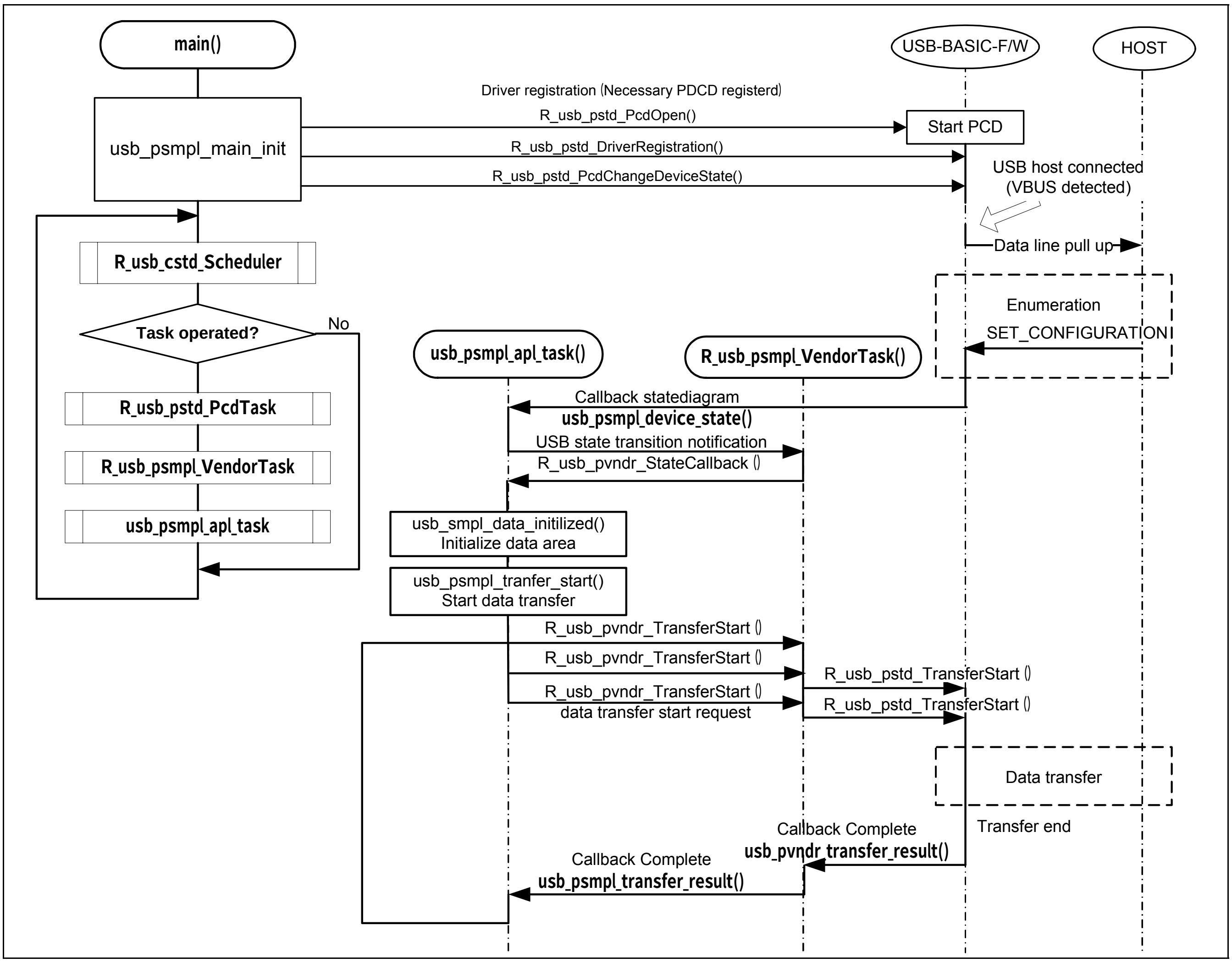

Figure 5.4 Application Operation Outline

5.3 Data Transfer

The data transfer is the customer-specific function specification and a transfer method, communication start or end timing, and buffer structure need to be changed based on a system.

5.3.1 Basic specification

As for the USB-BASIC-F/W, the data transfer is possible using the user buffer specified by *usb_utr_t* structure in Table 6-3. When the data transfer ends, the USB-BASIC-F/W sets *PID = NAK* and notifies the transfer end by the callback function.

The USB-BASIC-F/W updates the pipe status (data toggle) to pipe status (*utr_table.pipectr*) specified when the data transfer is demanded. Moreover, the pipe status (data toggle) is notified by the callback of the data transfer end. Therefore, because UPL memorizes the pipe status, the data transfer of "the control is exclusively possible" plural endpoint to which the data transfer is not done at the same time with one pipe is possible. The pipe status however should initialize at "*DATA0*" by factors of USB reset, the STALL release, the SET_CONFIGURATION request, and the SET_INTERFACE request, etc.

The size of the max packet of the Bulk pipe is a setting of 64 byte fixation.

5.3.2 Data Transfer Request

Use *R_usb_pstd_TransferStart()* to start the data transfer.

5.3.3 Notification of Transfer Result

The data transfer end is notified to the UPL using the callback function specified in the *usb_utr_t* structure. Refer to Table 6-7 for the content to be notified.

5.3.4 Note on Data Transmission

No note on data transmission

5.3.5 Notes on Data Reception

(1) Use a transaction counter for the received pipe.

When a short packet is received, the expected remaining receive data length is stored in *tranlen* of *usb_utr_t* structure and this transfer ends. When the received data exceeds the buffer size, data read from the FIFO buffer up to the buffer size and this transfer ends. When the user buffer area is insufficient to accommodate the transfer size, the *usb_cstd_forced_termination()* function may clear the receive packet.

(2) Received callback

When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated. When receiving a short packet or the data size is satisfied, the USB-BASIC-F/W judges the transfer end and generates the callback.

Example

When the data size of the reception schedule is 128 bytes and the maximum packet size is 64 bytes:

1 to 63 bytes received	A received callback is generated.
64 bytes received	A receive callback is not generated.
65 to 128 bytes received	A receive callback is generated.

5.3.6 Data Transfer Outline

Necessary information is set for the *usb_utr_t* structure and call *R_usb_pstd_TransferStart()*. An example of the data transfer is shown below.

```
void usb_pvndr_transfer_start( uint16_t pipe )
{
  g_usb_PsmplTrnMsg[pipe].pipenum    = pipe;
  g_usb_PsmplTrnMsg[pipe].tranadr    = g_usb_PsmplTrnPtr[pipe];
  g_usb_PsmplTrnMsg[pipe].tranlen    = g_usb_PsmplTrnSize[pipe];
  g_usb_PsmplTrnMsg[pipe].pipectr    = g_usb_PsmplPipeCtr[pipe];
  g_usb_PsmplTrnMsg[pipe].setup      = USB_NULL;
  g_usb_PsmplTrnMsg[pipe].complete   = (usb_cb_t)&usb_pvndr_transfer_result;
  R_usb_pstd_TransferStart((usb_utr_t *)&g_usb_PsmplTrnMsg[pipe]);
}
```

An example of the callback function (the transfer end is notified to the UPL via a message) is shown below.

```
void usb_pvndr_transfer_result(usb_utr_t *mess)
{
  usb_er_t      err;

  mess->msginfo = USB_SMPL_TRANSFER_END;

  err = R_USB_SND_MSG(USB_PVEN_MBX, (usb_msg_t*)mess);
  if( err != USB_E_OK )
  {
   while(1);
  }
}
```

5.4 Pipe Information

The pipe setting suited to this class driver needs to be created as the pipe information table. The pipe information of a vendor class driver is described in *uint16_t g_usb_psmpl_EpTbl1[]* of the *r_usb_vendor_descriptor.c* file.

5.4.1 Pipe Information Table

A pipe information table comprises the following four items (uint16_t × 4).

1. Pipe window select register (address 0x64)
2. Pipe configuration register (address 0x68)
3. Pipe maximum packet size register (address 0x6C)
4. Dummy data (not possible to delete)

5.4.2 Pipe Definition

The pipe information table structure used in the peripheral sample program is shown below. The macros are defined in the *r_usb_cstd_defusbip.h* file and refer to the header file by the pipe definition item of the pipe information table for the assignable value.

Structure example of pipe information table:

```
uint16_t g_usb_psmpl_EpTbl1[] =                        ← Pipe information table
{
    USB_PIPE4,                                         ← Pipe definition item 1
    USB_BULK | USB_BFREOFF | USB_DBLBON | USB_SHTNAKON | USB_DIR_P_IN  | USB_EP4,
                                                       ← Pipe definition item 2
    USB_MAX_PACKET(64),                                ← Pipe definition item 3
    USB_NULL,                                          ← Dummy data
            :
    USB_PDTBLEND,
}
```

(1). Pipe definition item 1: Specify the values to be set in the pipe window select register.
Pipe select: Specify the selected pipes (USB_PIPE4 to USB_PIPE7).

(2). Pipe definition item 2: Specify the values to be set in the pipe configuration register.

Transfer type	:	Specify either USB_BULK or USB_INT
BRDY operation designation	:	Specify USB_BFREOFF
Double buffer mode	:	Specify either USB_DBLBON or USB_DBLBOFF
SHTNAK operation designation	:	Specify either USB_SHTNAKON or USB_SHTNAKOFF
Transfer direction	:	Specify either USB_DIR_P_OUT or USB_DIR_P_IN
Endpoint number	:	Specify the endpoint number (EP1 to EP15) to the pipe

- The settable values differ depending on the pipes for the transfer type. For details, refer to the HM.
- Describe the pipe information according to the endpoint descriptor.
- Set USB_SHTNAKON for the receive direction pipe (USB_DIR_P_OUT).

(3). Pipe definition item 3: Specify the maximum packet size of the endpoint.
- Specify the maximum packet size: Set the value based on the USB specification.
- Specify the maximum packet size of the endpoint.

(4). Others.
- The pipe information is necessary for the number of endpoints that can be communicated simultaneously.
- Synchronize communication each transfer in the UPL.
- Write USB_PDTBLEND at the end of the table.
- Register the pipe information table using the *R_usb_pstd_DriverRegistration()* function.
- When the SET_CONFIGURATION request is received, set the pipe information to a register in the USB-BASIC-F/W.
- The pipe information does not support for the interface alternate setting.

5.5 Descriptor Information

It is necessary to create a descriptor according to the customer system. In the peripheral sample program, a sample table of a descriptor is described in *r_usb_vendor_descriptor.c* file.
The descriptor definitions comprise the following three types.

1. Standard Device Descriptor
   - uint8_t g_usb_psmpl_DeviceDescriptor[]
2. Configuration/Other_Speed_Configuration/Interface/Endpoint
   - uint8_t g_usb_psmpl_ConfigurationF_1[]
3. String Descriptor
   - uint8_t g_usb_psmpl_StringDescriptor0[]
   - uint8_t g_usb_psmpl_StringDescriptor1[]
   - uint8_t g_usb_psmpl_StringDescriptor2[]
   - uint8_t g_usb_psmpl_StringDescriptor3[]
   - uint8_t g_usb_psmpl_StringDescriptor4[]
   -

(1) ID registration

Set a vendor ID and product ID expanded to a descriptor.
Example) Vendor ID = 0x0000, product ID = 0x00FF

```
#define   USB_VENDORID          (0x0000u)          /* Vendor ID */
#define   USB_PRODUCTID         (0x00FFu)          /* Product ID */
```

(2) Device information

The device information is different depending on the selected speed.

```
#ifdef  USB_LSPERI_PP
        #define USB_PVDR_BLENGTH   32         /* Low Speed (PIPE 6-7) */
        #define USB_DCPMAXP        (8u)       /* DCP max packet size */
        #define USB_EPNUMS         (2)        /* Endpoint number */
        #define USB_INTEPMAXP      (8u)       /* Interrupt pipe max packet size *
/#endif  /* USB_LSPERI_PP */
#ifdef  USB_FSPERI_PP
        #define USB_PVDR_BLENGTH   46         /* Full Speed (PIPE 4-7) */
        #define USB_DCPMAXP        (64u)      /* DCP max packet size */
        #define USB_EPNUMS         (4)        /* Endpoint number */
        #define USB_INTEPMAXP      (64u)      /* Interrupt pipe max packet size */
#endif  /* USB_FSPERI_PP */
```

(3) Other information

Set the following information expanded to a descriptor.

```
#define   USB_BCDNUM        (0x0200u)        /* bcdUSB */
#define   USB_RELEASE       (0x0100u)        /* Release Number */
#define   USB_CONFIGNUM     (1u)             /* Configuration number */
```

Notes

1. For more details of each descriptor, refer to Chapter 9 of USB Revision 2.0 specification.
2. When changing a descriptor definition, change the pipe information table (sample table described in *r_usb_vendor_descriptor.c*) according to an endpoint descriptor.
3. Specify the serial number starting from 0 for the interface number.

5.6 In Order to Operate the USB-BASIC-F/W by the Peripheral mode

This chapter describes a procedure to operate the USB-BASIC-F/W by the peripheral mode as an example of the sample code.

5.6.1 Select a device

Table 5-1 lists the integrated development environment of each device provided by the USB-BASIC-F/W and hardware resources. Use a corresponding folder to a device to be used in the Project folder for each device. Please change the H/W resource folder name of the board used from "*HwResourceForUSB_G1C _ **board name***" to "*HwResourceForUSB_G1C*" when MCU is RL78.

Table 5-1 Hardware Resource of Sample Code

Device	Integrated development environment	Peripheral	Data rate	Hardware Resource Folder
R8C/3MU, R8C/34U, R8C/3MK, R8C/34K	HEW	√ √ √ √	FullSpeed FullSpeed FullSpeed FullSpeed	R8C\HwResourceForUSB
RL78/G1C	CubeSuite+/ EWRL78	√	FullSpeed LowSpeed	RL78\HwResourceForUSB_G1C_RSK
		√	FullSpeed LowSpeed	RL78\HwResourceForUSB_G1C_EVA

5.6.2 User Definition Information

Rewrite the user definition information file (*r_usb_usrconfig.h*) in the "*inc*" folder to set the function of the USB-BASIC-F/W. Settable items and outline in the user definition information file are shown below.

(1). Specify an operating mode

Set an operating mode of the USB module.
#define USB_FUNCSEL_PP USB_PERI_PP: Operate in peripheral mode

(2). Specify the data transfer rate

Set the data transfer rate of the USB communication. Make the macro in operation effective.
// #define USB_LSPERI_PP LowSpeed peripheral device
#define USB_FSPERI_PP FullSpeed peripheral device

(3). Specify the battery charging operation (Only RL78/USB)

Set the battery charging operation. Make the macro in operation effective.
#define USB_PERI_BC_ENABLE Enable batetry charging

The operation mode of USB-BASIC-F/W is specified with not the header file but the project file of the integration environment (build configuration of HEW or build mode of CubeSuite+ or Configurations of EWRL78).

A set content can be confirmed in the project file of CubeSuite+ according to the following procedures.

(1) Double click "*\Project\RL78\RL78G1C_48pin.mtpj*" to start the CubeSuite+.
(2) Open a property in CA78K0R (build tool) of a project tree.
(3) Click a table of a compile option.
(4) Items of the definition macro are defined as follows:

Definition macro	definition macro [2]
[0]	USB_FUNCSEL_PP=USB_PERI_PP
[1]	RL78USB

A set content can be confirmed in the project file of HEW according to the following procedures.

(1) Double click "*Project\R8C\R8C34K.hws*" to start the HEW.
(2) Select build to open "*Renesas M16C Standard Toolchain*"
(3) Open "*category: source*" and "*option: identifier definition*".
(4) Items of identifier definition are defined as follows:

Define	Value
USB_FUNCSEL_PP	USB_PERI_PP
R8CUSB	

A set content can be confirmed in the project file of EWRL78 according to the following procedures.

(1) Double click "*RL78G1C_48pin.eww*" to start the EWRL78.
(2) Click "*Project*" and "*Options*".
(3) Click a table of Preprocessor in C/C+compiler categry.
(4) Items of the defined symbol are defined as follows:
USB_FUNCSEL_PP=USB_PERI_PP
RL78USB

5.6.3 Setting Example of Peripheral Operation

Setting procedure to operate the USB-BASIC-F/W as the peripheral mode is shown below using the CubeSuite+ as an example.

<Setting procedure>

1. Select a hardware resource
   Change the folder name of "*HwResourceForUSB_RL78_G1C_* ***board name***" to "*HwResourceForUSB_G1C*".
2. Set a workspace
   Double click "*Project\RL78\RL78G1C_48pin.mtpj*" and to start the CubeSuite+. Then select "*Peripheral*" in build mode of the workspace.
3. Create an execute file
   Select "*build*" and "*rebuild project*" from the upper tab of the workspace to build.
4. Set the debug environment
   Select a debug tool to use in a debug tool of a project.
5. Connect with the evaluation board
   Select "*debug*" and "*connect*" from the upper tab of the workspace to start the debug environment.
6. Download an execute file
   Select "*debug*" and "*download to debug tool*" from the upper tab of the workspace to download the execute file to the debug environment.
7. Execute an execute file
   Select "*debug*" and "*execute after reset*" from the upper tab of the workspace to execute a program.

Setting procedure to operate the USB-BASIC-F/W as the peripheral mode is shown below using the HEW as an example.

< Setting procedure >

1. Set a workspace
   Double click "*Fw.hws*" to start the HEW. Then select "*PERI*" in the configuration of the workspace.
2. Create an execute file
   Select "*build*" and "*rebuild project*" from the upper tab of the workspace to build.
3. Set the debug environment
   Select a debug tool to use in a debug tool of a project.
4. Connect with the evaluation board

Select "*debug*" and "*connect*" from the upper tab of the workspace to start the debug environment.

5. Download an execute file
   Select "*debug*" and "*download to debug tool*" from the upper tab of the workspace to download the execute file to the debug environment.

6. Execute an execute file
   Select "*debug*" and "*execute after reset*" from the upper tab of the workspace to execute a program.

Setting procedure to operate the USB-BASIC-F/W as the peripheral mode is shown below using the EWRL78 as an example.

<Setting procedure>

1. Select a hardware resource
   Change the folder name of "*HwResourceForUSB_RL78_G1C_* ***board name***" to "*HwResourceForUSB_G1C*".

2. Set a workspace
   Double click "*RL78G1C_48pin.eww*" and to start the EWRL78. Then select "*Peripheral*" from the upper pull down menu of the workspace.

3. Create an execute file
   Select "*Project*" and "*Make*" from the upper tab of the workspace to build.

4. Set the debug environment
   Select "*Project*" and "*Options*". Then select a debug tool to use at Setup tab in Debugger category.

5. Connect with the evaluation board and Download an execute file
   Select "*Project*" and "*Download and Debug*" from the upper tab of the workspace to download the execute file to the debug environment.

6. Execute an execute file
   Select "*Debug*" and "*Go*" from the upper tab of the workspace to execute a program.

5.6.4 Change the USB-BASIC-F/W

The program and header file shown below need to be changed in order to operate the USB-BASIC-F/W.
Sample functions for the Renesas USB MCU will be provided. Change them according to the user system.

- Initializes the MCU for control, interrupt handler, interrupt control, and etc. Refer to

Table 5-2.

- Time adjustment of specified time wait function (*usb_cpu_delay_xms()* function, *usb_cpu_delay_1u()* function)
  Generate the specified wait time using loop processing. Adjust for the time in order to generate the specified time, such as changing the number of loops according to the system.
- Set the function to disable and enable the USB associated interrupts in order to use the scheduler function.
  (*usb_cpu_int_disable()* function, *usb_cpu_int_enable()* function)

The USB interrupt disable function (*usb_cpu_int_disable()* function) disables the USB interrupt and the USB interrupt enable function (*usb_cpu_int_enable()* function) enables the USB interrupt for the USB-BASIC-F/W. Perform the setting according t

Table 5-2 Function List

Type	Function Name and argument	Description	Notes
void	usb_cpu_mcu_initialize(void)	MCU initialization (oscillation control, etc.)	
void	usb_cpu_target_init(void)	System initialization	
void	usb_cpu_set_pin_function (uint16_t function)	USB function setting of the MCU(pin setting, etc.)	
void	usb_cpu_usb_interrupt (void)	USB interrupt handler	
void	usb_cpu_usbint_init (void)	USB interrupt enabled	
void	usb_cpu_int_enable(void)	USB interrupt enabled for the scheduler	
void	usb_cpu_int_disable(void)	USB interrupt disabled for the scheduler	
void	usb_cpu_delay_1us(uint16_t time)	1 μs wait processing	
void	usb_cpu_delay_xms(uint16_t time)	1 ms wait processing	
void	usb_cpu_stop_mode(void)	Execute the STOP instruction	

5.6.5 User System Definition Information File (*r_usb_usrconfig.h*)

Register the number of String Descriptors.

```
#define USB_STRINGNUM        (7u)                /* Max of string descriptor */
```

6. Peripheral Controller Driver (PCD)

6.1 Basic Function

The PCD is a program to control the hardware when operating target devices as USB functions. The USB-BASIC-F/W analyzes requests issued from the UPL and controls the hardware. The hardware control result is notified to the UPL using the return value or callback function. Requests from the hardware are informed using the callback function of the driver information registered in the USB-BASIC-F/W. Start the USB-BASIC-F/W shown in chapter 6.2.1 and register the UPL shown in chapter 6.2.3 to make USB-BASIC-F/W as a peripheral device.

Functions of the PCD are shown below.

1. Detection for the USB state change with the connected host and notification for the change result: Chapter 6.2.3
2. Enumeration with the connected host: Chapter 6.2.7
3. Notification for a USB request: Chapter 6.2.4
4. Data transfer and notification for transferred result: Chapter 6.2.5
5. USB state control (USB state control and notification for control result): Chapter 6.2.6

6.2 Operation Outline

6.2.1 Starting the PCD

Start the USB-BASIC-F/W using API function *R_usb_pstd_PcdOpen()*.

6.2.2 Registration for the UPL

The UPL registers the information in Table 6-1 to the USB-BASIC-F/W using API function *R_usb_pstd_DriverRegistration()*

USB-BASIC-F/W preserves information in global variable (*g_usb_PcdDriver*).

```
typedef struct
{
  uint16_t        *pipetbl;      /* Pipe Define Table address */
  uint8_t         *devicetbl;    /* Device descriptor Table address */
  uint8_t         *configtbl;    /* Configuration descriptor Table address */
  uint8_t         **stringtbl;   /* String descriptor Table address */
  usb_cb_info_t   statediagram;  /* Device status */
  usb_cb_trn_t    ctrltrans;     /* Control Transfer */
}usb_pcdreg_t;
```

Table 6-1 Member of usb_pcdreg_t Structure

Members	Functions	Notes
*pipetbl	Register an address of pipe information table.	
*devicetbl	Register an address of Device Descriptor table.	
*configtbl	Register an address of Configuration Descriptor table.	
**stringtbl	Register an address of String Descriptor address table.	
statediagram	Register a function to start when the USB state is transited.	
ctrltrans	Register a function to start when a class request or vendor request is generated.	

6.2.3 Notification for USB State Change

To notify UPL the USB state transition etc, the USB-BASIC-F/W executes USB state transition callback function (**g_usb_PcdDriver.statediagram*) registered in USB-BASIC-F/W. The USB-BASIC-F/W notifies the information below to the UPL using the second argument of the callback function. Analyze the USB state and perform suitable processing to the system.

USB state transition

USB_ STS_DETACH:	Detach detection
USB_ STS_ATTACH:	Attach detection
USB_ STS_DEFAULT:	Default state transition (USB bus reset detection)
USB_ STS_ADDRESS:	Address state transition (*Set_Address* request reception)
USB_ STS_CONFIGURED:	Configured state transition (*Set_Configuration* request reception)
USB_ STS_SUSPEND:	Suspend state transition (suspend detection)
USB_ STS_RESUME:	Suspend state cancellation (resume detection)
USB_PORTENABLE:	Pull up the D+ (RL78/USB contains the case where "Pull up D-" is.)

6.2.4 Control Transfer Notification

The USB-BASIC-F/W automatically returns a standard request and enumerate to USB host in Chapter 6.2.7. When a device class (vendor class) request is received, the control transfer callback function (**g_usb_pstd_Driver.ctrltrans*) registered in the USB-BASIC-F/W is executed. The USB-BASIC-F/W notifies UPL the information in Table 6-2using the first argument of the callback function. Analyze a USB request and perform processing based on the UPL. When receiving at the following standard requests, the control transfer callback function is executed.
When receiving *Get_Descriptor request* and *bRecipient* is an interface.
When receiving *Clear_Feature* request or *Set_Feature* request.

The following request types are notified by the second argument when the call backing is done by a standard request.

USB_CLEARSTALL	Receive *Clear_Feature* request (Clear STALL)
USB_CLEARREMOTE	Receive *Clear_Feature* request (Disable remote wakeup)
USB_SETREMOTE	Receive *Set_Feature* request (Enable remote wakeup)
USB_SETSTALL	Receive *Set_Feature* request (Set STALL)
USB_RECIPIENT	Receive *Get_Descriptor* request and bRecipient is an interface

```
typedef struct
{
   union {
      struct {                /* Characteristics of request */
         uint8_t  bRecipient:5;     /* Recipient */
         uint8_t  bType:2;          /* Type */
         uint8_t  bDirection:1;     /* Data transfer direction */
         uint8_t  bRequest:8;       /* Specific request */
      } BIT;
      uint16_t wRequest;      /* Control transfer request */
   } WORD;
  uint16_t     wValue;        /* Value */
  uint16_t     wIndex;        /* Index */
  uint16_t     wLength;       /* Length */
} usb_request_t;
```

Table 6-2 Member of usb_request_t Structure

Members	Functions	Notes
wRequest	The value is wRequest of request. (The value is BREQUEST of USBREQ register.) The bit can refer for wRequest in a union type.	
wValue	The value is wValue of request. (The value is USBVAL register.)	
wIndex	The value is wIndex of request. (The value is USBINDEX register.)	
wLength	The value is wLength of request. (The value is USBLENG register.)	

6.2.5 Issuing the Transfer Request for USB-BASIC-F/W

Please assume the following structures to be an argument and call an API function *R_usb_pstd_TransferStart()*, when the UPL wants to the data transfer. The USB-BASIC-F/W preserves address information of the argument in global variable (*g_usb_LibPipe*). Therefore, please maintain the realities of the argument in UPL until the data transfer ends.

```
struct usb_utr_t
{
  usb_strct_t   msginfo;       /* Message Info for F/W */
  usb_strct_t   pipenum;       /* Pipe number */
  usb_strct_t   status;        /* Transfer status */
  usb_strct_t   flag;          /* Flag */
  usb_cb_t      complete;      /* Call Back Function Info */
  uint8_t       *tranadr;      /* Transfer data Start address */
  uint16_t      *setup;        /* Setup packet(for control only) */
  uint16_t      pipectr;       /* Pipe control register */
  usb_leng_t    tranlen;       /* Transfer data length */
  uint8_t       dummy;         /* Adjustment of the byte border */
}
```

Table 6-3 Member of usb_utr_t Structure

Members	Functions	Notes
msginfo	This member is message information that USB-BASIC-F/W uses. It is set using the API function.	
pipenum	Specify the pipe number in the UPL.	
status	The USB-BASIC-F/W returns the following status information. USB_DATA_OK: Data transfer (transmission/reception) normal end USB_DATA_SHT: Data reception normal end with less than specified data length USB_DATA_OVR: Receive data size exceeded USB_DATA_ERR: No-response condition or over/under run error detected USB_DATA_DTCH： Detach detected USB_DATA_STALL: STALL or Max packet size error detected USB_DATA_STOP: Data transfer forced end	
complete	Specify the callback function to be executed in the UPL at the data transfer end. Type declaration is as follows: typedef void (*usb_cb_t)(usb_utr_t*);	
*tranadr	Specify the following information in the UPL. Reception: Buffer address to store the receive data Transmission: Buffer address to store the transmit data To secure the bigger area than the data length at the specified with tranlen.	
pipectr	Specify the PIPExCTR register information in the UPL. Control the sequence bit of DATA0/DATA1 according to bit 6 of the applicable member. Set USB_NULL for the initial state and the returned value by the USB-BASIC-F/W after the second communication. USB-BASIC-F/W returns the PIPExCTR register information.	
tranlen	Specify the following information in the UPL. Reception: Data length to be received Transmission: Data length to be transmitted The maximum length that can be sent and received is 65535 bytes. USB-BASIC-F/W stored the remaining transmit/receive data length after the end of data transfer.	
Others	Not used	

6.2.6 Request and Notify to Change the USB State for the USB-BASIC-F/W

To call API function *R_usb_pstd_PcdChangeDeviceState()*, when the UPL wants to change the USB state. Specify the controlled descriptions using the API function argument. Perform processing using the context which called the API function and notify its result.

Information controlled by the USB-BASIC-F/W can be obtained using API function *R_usb_pstd_DeviceInformation()*.

6.2.7 Enumeration

The USB-BASIC-F/W automatically returns standard requests from the USB host. Supported standard requests by USB-BASIC-F/W are as follows.

(1) GET_DESCRIPTOR

(2) SET_ADDRESS

(3) SET_CONFIGURATION

(4) GET_STATUS

(5) GET_CONFIGURATION

(6) GET_INTERFACE

(7) CLEAR_FEATURE

(8) SET_FEATURE

(9) SET_INTERFACE

When the USB-BASIC-F/W is configured to the host (transition of configured state), the USB-BASIC-F/W notifies the configuration to the UPL using the registered callback function (**g_usb_PcdDriver.statediagram*) for the USB state transition. Analyze the USB state of the second argument and perform processing suited to a system for the UPL. The sample application initializes the global variable at the transition of USB_STS_CONFIGURED state to enable the data transfer.

There is a time lag in the acquisition timing of stage transition information and request information in the control transfer of the USB-BASIC-F/W. Therefore, the disagreement between stage information and the request might occur when the new control transfer is begun before request information is acquired.

6.2.8 Peripheral Battery Charging control (PBC)

PBC is the H/W control program for the target device that operates the Charging Port Ditection (CPD from now on and description) defined by USB Battery Charging Specification Revision 1.2.

The CPD is immediately executed after the USB-BASIC-F/W notifies the USB state transition (USB_STS_ATTACH) to UPL by the callback function. The USB-BASIC-F/W notifies the result of the CPD to UPL by the callback function of USB state transition (USB_PORTENABLE) using the first argument.

The result value (using the first argument of callback) of the CPD notified to UPL is as follows.

0 : Standard Downstream Port (SDP) Detection

1 : Charging Downstream Port (CDP) Detection

2 : Dedicated Charging Port (DCP) Detection

The processing flow of PBC is shown Figure 6.1.

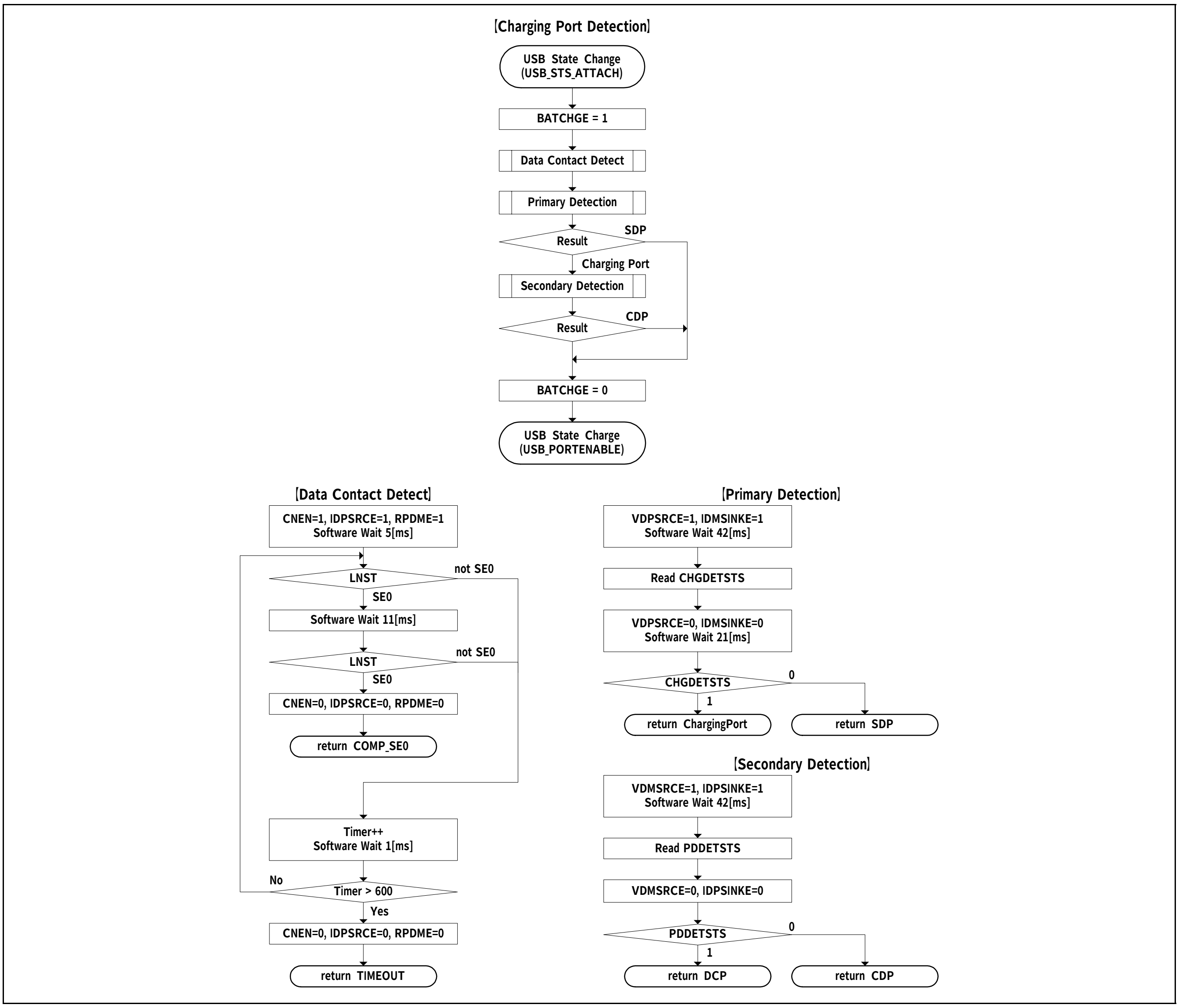

Figure 6.1 PBC processing flow

6.2.9 Notes on the USB-BASIC-F/W

Even if the suspend state is generated, the USB-BASIC-F/W doesn't interrupt the data transfer

The USB-BASIC-F/W stops the data transfer when detecting detach.

USB-BASIC-F/W doesn't correspond to the multi configuration setting.

USB-BASIC-F/W doesn't correspond to the interface alternates setting (The pipe register cannot be changed).

The USB-BASIC-F/W includes the following functions. For more details, refer to Chapter 6.3.

(1) Enable and disable the USB port.

(2) Change the USB state (remote wakeup).

(3) Stall the pipe.

(4) Stop the PCD.

(5) Access the FIFO buffer for the Control transfer.

6.3 PCD API Function

Request the hardware control from the UPL using the API function. The API function is in the *r_usb_pdriverapi.c* file. When using the PCD API function, follow the order shown in Table 6-4 to include the header file. Table 6-5 lists the PCD API function.

Table 6-4 List of PCD API header file

File Name	Description	Notes
r_usb_ctypedef.h	Variable type definition	
r_usb_ckernelid.h	System header file	
r_usb_cdefusbip.h	Various definition for the USB driver	
r_usb_api.h	USB driver API function definition	

Table 6-5 List of PCD API Function

Function Name	Description	Notes
R_usb_pstd_PcdTask	PCD task	
R_usb_pstd_PcdOpen	PCD task activation (PCD task initialization)	
R_usb_pstd_DriverRegistration	UPL registration	
R_usb_pstd_TransferStart	Data transfer execution request	
R_usb_pstd_TransferEnd	Data transfer forced end request	
R_usb_pstd_PcdChangeDeviceState	USB device state change request	
R_usb_pstd_DeviceInformation	Obtaining the USB device information	
R_usb_pstd_SetStallPipe0	Setting PID of pipe 0 to STALL	
R_usb_pstd_SetPipeStall	Setting PID of pipe4,5,6,and 7 to STALL	
R_usb_pstd_ControlRead	FIFO access execution request for control read transfer	
R_usb_pstd_ControlWrite	FIFO access execution request for control write transfer	
R_usb_pstd_ControlEnd	Control transfer end request	
R_usb_pstd_SetPipeRegister	Set pipe information to the register.	

6.4 PCD Callback function

The USB-BASIC-F/W notifies the USB state change and data transfer end to the UPL using the callback function. When a driver is registered and the API function is called, the UPL specifies the callback function. When making the newly callback function, follow the order shown in Table 6-4 to include the header file in the same way when using the API function. Moreover, the PCD callback function list is shown in Table 6-6

Table 6-6 List of PCD callback Function

Function Name	Description	Notes
*g_usb_PcdDriver.statediagram	The USB state transition is detected	
*g_usb_PcdDriver.ctrltrans	The control transfer is occurred	
* g_usb_LibPipe[pipe]->complete	The data transfer is occurred	

6.5 Details for the API Function and Callback function

Details for the API function and callback function are shown below.

R_usb_pstd_PcdTask

PCD task

Format

void R_usb_pstd_PcdTask(void)

Arguments

— —

Return Value

— —

Description

To call *usb_pstd_pcd_task()* function.

The *usb_pstd_pcd_task()* function is executed responded processing.

- Return the USB standard request.
- When detecting a class request and vendor request, call the control transfer callback function registered by the UPL.
- When detecting the USB state transition, call the USB state transition callback function registered by the UPL. Perform processing requested via the API function.

The *usb_pstd_pcd_task()* function perform the data transfer requested via the API function.

- When the data transfer ends, call the callback function specified using the API function.

Notes

1. Call this function in a loop where scheduler processing is performed.
2. Deadlock processing with while(1) when receiving the invalid message.

Example

```
void main(void)
{
  usb_psmpl_main_init();
  while( 1 )
  {
   if(R_usb_cstd_Scheduler() == USB_FLGSET )
   {
      R_usb_pstd_PcdTask();
      usb_psmpl_apl_task();
   }
  }
}
```

R_usb_pstd_PcdOpen

PCD task start

Format

void R_usb_pstd_PcdOpen(void)

Arguments

— —

Return Value

— —

Description

Initialize global variables.

Note

1. To call the starting of USB-BASIC-F/W.

Example

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

  /* USB-IP initialized */
  R_usb_pstd_PcdChangeDeviceState(USB_DO_INITHWFUNCTION)

  /* PCD driver open & registration */
  R_usb_pstd_PcdOpen();                     /* PCD task open */
  usb_psmpl_driver_registration();          /* Sample driver registration */

  /* USB-IP is set to the peripheral */
  R_usb_pstd_PcdChangeDeviceState(USB_DO_SETHWFUNCTION);
}
```

R_usb_pstd_DriverRegistration

Peripheral device class driver (PDCD) registration

Format

void R_usb_pstd_DriverRegistration(usb_pcdreg_t *registinfo)

Argument

*registinfo Class driver structure

Return Value

— —

Description

Register the UPL to the USB-BASIC-F/W.

Notes

1. Call this function using the UPL for initialization.
2. There is only one registerable driver. Refer to Chapter 6.2.1 for registered information.

Example

```
void usb_psmpl_driver_registration(void)
{
  usb_pcdreg_t  driver;

  /* Driver registration */
  driver.pipetbl      = g_usb_psmpl_EpTbl1;
  driver.devicetbl    = g_usb_psmpl_DeviceDescriptor;
  driver.configtbl    = g_usb_psmpl_ConfigurationF_1;
  driver.stringtbl    = g_usb_psmpl_StringPtr;
  driver.statediagram = &usb_apl_change_device_state;
  driver.ctrltrans    = &usb_psmpl_control_transfer;
  R_usb_pstd_DriverRegistration(&driver);
}
```

R_usb_pstd_TransferStart

Data transfer request

Format

usb_er_t R_usb_pstd_TransferStart(usb_utr_t * utr_table)

Argument

* utr_table	Data transfer structure

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap (The pipe is using.)

Description

Request the data transfer of each pipe. When the specified data size is satisfied, receiving a short packet, and an error occurs, the data transfer ends.
When the data transfer ends, call the callback function of the argument in the structure member. Remaining data length of transmission and reception, status, and information of transfer end are set in the argument of this callback function (*utr_table*).
When the data transfer is restarted with the same pipe, it is necessary to put the pipe status that does the communication demand this time as the pipe status (data toggle) that did the forwarding end last time together. Structure member (*utr_table.pipectr*) of the argument must set the pipe status. When the factors of USB reset or the clear STALL, etc. are generated, the pipe status should be initialized at "*DATA0*".
When a transfer start request is generated to the pipe during the data transfer, return USB_E_QOVR.

Notes

1. Refer to Table 6-3 Member of usb_utr_t Structure for the structure of data transfer.
2. This function does not support for the control transfer.
3. When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated.

Example

```
usb_utr_t   g_usb_PsmplTrnMsg[USB_TBL_MAX];
void usb_pvndr_data_transfer(usb_pipe_t pipe)
{
  /* PIPE Transfer set */
  g_usb_PsmplTrnMsg[pipe].pipenum = pipe;
  g_usb_PsmplTrnMsg[pipe].tranadr = g_usb_PsmplTrnPtr[pipe];
  g_usb_PsmplTrnMsg[pipe].tranlen = g_usb_PsmplTrnSize[pipe];
  g_usb_PsmplTrnMsg[pipe].pipectr = g_usb_PsmplPipeCtr[pipe];
  g_usb_PsmplTrnMsg[pipe].setup   = USB_NULL;
  g_usb_PsmplTrnMsg[pipe].complete   = (usb_cb_t)&usb_pvndr_transfer_result;
  R_usb_pstd_TransferStart((usb_utr_t *)&g_usb_PsmplTrnMsg[pipe]);
}
```

R_usb_pstd_TransferEnd

Data transfer forced end request

Format

usb_er_t R_usb_pstd_TransferEnd(usb_pipe_t pipe, usb_strct_t_t msginfo)

Arguments

pipe	Pipe number
msginfo	Communication status

Return Value

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

Description

Set the following values to argument *msginfo* and request forced end of the data transfer to the USB-BASIC-F/W.

- USB_DO_TRANSFER_STP: Data transfer forced end (The PCD calls back.)
- USB_DO_TRANSFER_TMO: Data transfer timeout (The PCD does not call back.)

The transfer end is notified using the callback function set when the data transfer is requested (R_usb_pstd_TransferStart) for the forced end due to *msginfo=USB_DO_TRANSFER_STP*. The remaining data length of transmission and reception, pipe control register value, and transfer s*tatus = USB_DATA_STOP* are set using the argument of the callback function (usb_utr_t). When the forced end request to the pipe that doesn't execute data transfer is generated, USB_E_QOVR is returned.

Notes

1. When data transmission is suspended, the FIFO buffer of the SIE is not cleared.
   When the FIFO buffer is transmitted in the double buffer, the data that has not been transmitted yet may be remained in the FIFO buffer.
2. When argument pipes are pipe 0 to pipe 3, the USB_E_QOVR error is returned. The USB_E_ERROR error is returned for pipe 8 or more.

Example

```
void  usb_smp_task(void)
{
  R_usb_pstd_TransferEnd(USB_PIPE4, USB_DO_TRANSFER_STP);
}
```

R_usb_pstd_PcdChangeDeviceState

USB device state change request

Format

usb_er_t R_usb_pstd_PcdChangeDeviceState(usb_strct_t msginfo)

Argument

msginfo USB state to be changed

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

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

- USB_DO_PORT_ENABLE
  Perform the pull-up request (connection notification to a host) of the USB data line (D+/D- line).
- USB_DO_PORT_DISABLE
  Perform the pull-up request (cutoff notification to a host) of the USB data line (D+/D- line).
- USB_DO_REMOTEWAKEUP
  Request executing the remote wakeup.
- USB_DO_INITHWFUNCTION
  Start the USB-IP and perform the software reset. Execute this function before USB-BASIC-F/W starts.
- USB_DO_SETHWFUNCTION
  Set the the USB-IP as the USB peripheral device. Execute this function after registering UPL.

Notes

1. When a connection or disconnection is detected by an interrupt in USB-BASIC-F/W, USB-BASIC-F/W automatically releases the USB data lines pull up.
2. This applicable function processing is executed without the PCD task.

Example

```
void  usb_smp_task(void)
{
  R_usb_pstd_PcdChangeDeviceState(USB_DO_INITHWFUNCTION);
  R_usb_pstd_PcdOpen();                    /* PCD task open */
  usb_psmpl_driver_registration();         /* Sample driver registration */
  R_usb_pstd_PcdChangeDeviceState(USB_DO_SETHWFUNCTION);
    :
    :

}
```

R_usb_pstd_DeviceInformation

Obtaining the USB device state information

Format

void R_usb_pstd_DeviceInformation (uint16_t *table)

Argument

*table Table address where the obtained information is stored

Return Value

— —

Description

Obtain the USB device information. Store the following information to an address specified to the argument (**table*).
[0]: USB state (VBSTS and DVSQ field values in the INTSTS0 register)
[1]: Configuration number (wValue of SET_CONFIGURATION request)
[2]: Number of interfaces (g_usb_PcdDriver.configtbl[USB_CON_NUM_INTERFACE])
[3]: Remote wakeup flag (Enable: USB_YES, disable: USB_NO)

Notes

1. Prepare the area of 4word in argument **table*.

Example

```
void  usb_smp_task(void)
{
  uint16_t  res[4];
    :
   R_usb_pstd_DeviceInformation(res);
    :
}
```

R_usb_pstd_SetStallPipe0

STALL setting for the PID of Pipe 0 (for the control transfer)

Format

void	R_usb_pstd_SetStallPipe0(void)

Arguments

—	—

Return Value

—	—

Description

Set STALL to the PID of PIPE0.

Notes

1. Call this function when responses STALL by the class request or the vendor request.

Example

```
void usb_psmpl_control_transfer(usb_request_t *data1, uint16_t data2)
{
  if (data1->TypeRecip == USB_INTERFACE )
  {
    R_usb_pstd_SetStallPipe0();
  }
  else
  {
   usb_smpl_vendore_request(data1);
  }
}
```

R_usb_pstd_SetPipeStall

STALL setting for the PID of pipe x (for the data transfer)

Format

void	R_usb_pstd_SetPipeStall(usb_pipe_t pipe)

Argument

pipe	Pipe number

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

Set STALL to the PID of the pipe number specified by the argument.

Notes

1. Pipe 0 setting to the argument pipe is an error. Use the *R_usb_pstd_SetStallPipe0()* function.

Example

```
void  usb_smp_task(void)
{
    :
  R_usb_pstd_SetPipeStall(USB_PIPE4);
    :
}
```

R_usb_pstd_ControlRead

FIFO access request for control read transfer

Format

uint16_t R_usb_pstd_ControlRead (usb_leng_t bsize, uint8_t *table)

Argument

bsize	Transmit data buffer size
*table	Transmit data buffer address

Return Value

USB_WRITESHRT	Data write end (short packet data write)
USB_WRITING	Data write in progress (additional data present)
USB_FIFOERROR	FIFO access error

Description

This function is used during the data stage of the control read transfer. Read the data from an area the argument (**table*) shows and write it to the FIFO buffer.

USB-BASIC-F/W continues the data stage until transmits a short packet or generates the OUT token. When the specified data size is equal to the size of the max packet, the NULL packet is transmitted by the IN token after specified data is transmitted.

Note

1. Call this function at the data stage of the control read transfer.

Example

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques1(usb_request_t *data1, uint16_t data2)
{
  if (data1->TypeRecip == USB_INTERFACE )
  {
     R_usb_pstd_ControlRead(10,(uint8_t*)&g_usb_smp_buff);
  }
  else
  {
    R_usb_pstd_SetStallPipe0();
  }
}
```

R_usb_pstd_ControlWrite

FIFO access request for control write transfer

Format

void R_usb_pstd_ControlWrite(usb_leng_t bsize, uint8_t *table)

Argument

bsize	Receive data buffer size
*table	Receive data buffer address

Return Value

— —

Description

This function is used during the data stage of the control read transfer. Read the data from the FIFO buffer and write it to the area that the argument (**table*) shows.

Notes

1. Call this function at the data stage of the control write transfer.
2. Read the data up to the specified data length.
3. Although the received data is less than the data length, reading ends when a short packet is received.

Example

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques2(usb_request_t *data1, uint16_t data2)
{
  if (data1->TypeRecip == USB_INTERFACE )
  {
    R_usb_pstd_ControlWrite(10,(uint8_t*)&g_usb_smp_buff);
  }
  else
  {
   R_usb_pstd_SetStallPipe0();
  }
}
```

R_usb_pstd_ControlEnd

Control transfer end request

Format

void R_usb_pstd_ControlEnd(uint16_t status)

Argument

status Status

Return Value

— —

Description

This function is used during the data stage of the control transfer.
Set any of the following values to the argument (*status*).

- USB_CTRL_END
  Status stage normal end
- USB_DATA_STOP
  Return NAK to host at status stage.
- USB_DATA_ERR / USB_DATA_OVR
  Return STALL to a host at status stage.

Notes

1. Call this function at the status stage of the control transfer.
2. When specifying USB_CTRL_END to the argument (*status*), set PID = BUF and CCPL = 1.
3. When specifying USB_CTRL_END to the argument (*status*) while PID is STALL, STALL is returned.

Example

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques3(usb_request_t *data1, uint16_t data2)
{
  if (data1->TypeRecip == USB_INTERFACE )
  {
     R_usb_pstd_ControlEnd(USB_CTRL_END);
  }
  else
  {
     R_usb_pstd_ControlEnd(USB_DATA_ERR);
  }
}
```

R_usb_pstd_SetPipeRegister

Set the pipe information to the register

Format

void	R_usb_pstd_SetPipeRegister(uint16_t* table, uint16_t command)

Argument

table	Pipe information table
command	Command

Return Value

—	—

Description

- When the command is "USB_NO".
  All pipes specified with the pipe information table are unused set.
- When the command is "USB_YES".
  All pipes specified with the pipe information table are unused set.
  After it makes it to unused, the pipe information table follows and sets all pipes.

Notes

1. When the *Set_Configuration* request is received, USB-BASIC-F/W executes this processing.

Example

```
void usb_pstd_set_configuration3(void)
{

  if( g_usb_PcdRequest.TypeRecip == USB_DEVICE )
  {
          :
      if( g_usb_PcdConfigNum != (uint8_t)g_usb_PcdRequest.wValue )
      {
         /* Configuration number set */
         g_usb_PcdConfigNum   = (uint8_t)g_usb_PcdRequest.wValue;
         R_usb_pstd_SetPipeRegister(g_usb_PcdDriver.pipetbl, USB_NO);
      }
      if( g_usb_PcdConfigNum > 0 )
      {
         R_usb_pstd_SetPipeRegister(g_usb_PcdDriver.pipetbl, USB_YES);
      }
      return;
          :
  }
  R_usb_pstd_SetStallPipe0();

}
```

*g_usb_PcdDriver.statediagram

Callback when detecting the USB state transition

Format

void (*g_usb_PcdDriver.statediagram)((uint16_t)data1, (uint16_t)device_state);

Argument

data1	Normally not used, configuration number for Set_Configuration
device_state	USB state

Return Value

—	—

Description

USB state transition generation is notified to the UPL.

- Resume detection
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_RESUME) ;
- State transition interrupt detection
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_DEFAULT);
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_ADDRESS);
  (*g_usb_PcdDriver.statediagram)(g_usb_PcdConfigNum, USB_STS_CONFIGURED);
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_SUSPEND);
- Detach detection
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_DETACH);
- Attach detection
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_ATTACH);
- USB data line is set to pull up
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_PORTENABLE);

Notes

1. Communication speed of a device is not notified using callback when a reset is detected.
2. The PCD does not call back when the *Set_Configuration* request is received and the structure number is not changed.
3. The ADDRESS state is notified when the *Set_Configuration* request is received and the structure number is 0.

Example

Processing of the function that callback is done as an example.

```
void usb_apl_change_device_state(uint16_t data, uint16_t state)
  case USB_STS_CONFIGURED:     /* Device configured */
   configuratuion_num= (uint8_t)data;
   usb_psmpl_open();
  break;
  case USB_STS_ATTACH:         /* Device attach */
  break;
  case USB_STS_DETACH:         /* Device detach */
   configuratuion_num= (uint8_t)0;
  break;
  case USB_STS_SUSPEND:        /* Device suspend */
  case USB_STS_RESUME:         /* Device resume */
  break;
  case USB_STS_DEFAULT:        /* Device default */
  case USB_STS_ADDRESS:        /* Device addressed */
      configuratuion_num= (uint8_t)0;
  break;
  case USB_PORTENABLE:         /* D+ line pull up */
  break;
  default:
   usb_apl_dummy_function(data,state);
  break;
  }
}
```

*g_usb_PcdDriver.ctrltrans

Callback for control transfer

Format

void (*g_usb_PcdDriver.ctrltrans)((usb_request_t *)request, (uint16_t)data;

Argument

request	USB request
data	Stage of control transfer

Return Value

— —

Description

Generation for a class request or vendor request control transfer is notified to the UPL. The transfer stage category of the second argument is shown below. For more details, refer to the User's Manual: Hardware.

- USB_CS_IDST /* Idle or setup stage */
- USB_CS_RDDS: /* Control read data stage */
- USB_CS_WRDS: /* Control write data stage */
- USB_CS_WRND: /* Control write no data status stage */
- USB_CS_RDSS: /* Control read status stage */
- USB_CS_WRSS: /* Control write status stage */
- USB_CS_SQER: /* Control sequence error */

(*g_usb_PcdDriver.ctrltrans)((usb_request_t*)&g_usb_PcdRequest, (uint16_t)intseq);

When the standard requests shown below are received, generation for the class request or vendor request control transfer is notified to the UPL.

- When the Clear_Feature request is received and remote wakeup is cancelled :
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_CLEARREMOTE);
- When the Clear_Feature request is received and STALL of ENDPOINT is cancelled :
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_ CLEARSTALL);
- When the Get_Descriptor request is received and bRecipient in its request is USB_INTERFACE ;
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_RECIPIENT);
- When the Get_Interface request is received and it is an alternate notificaion request.
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_GET_INTERFACE);
- When the Set_Feature request is received and remote wakeup is enabled ;
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_SETREMOTE);
- When the Set_Feature request is received and stall of endpoint is set;
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_SETSTALL);

Notes

1. The USB-BASIC-F/W does not support for the interface alternate setting (pipes cannot be switched).

   When the *Clear_Feature* request is normally accepted, callback is notified to the UPL. Determine if STALL is cancelled for the pipe in which the UPL sets STALL.

2. The alternative notification demand of the *Get_Interface* request responds "0".

Example

Processing of the function that callback is done as an example.

```
void usb_psmpl_control_transfer(usb_request_t *request, uint16_t data)
{
  g_usb_SmplRequest   = *request;

  switch(g_usb_SmplRequest.wRequest & USB_BMREQUESTTYPETYPE)
  {
    case  USB_STANDARD:
       switch( data )
       {
          case USB_SETREMOTE:
             /* Enable Remote wakeup */
          break;
          case USB_CLEARREMOTE:
             /* Disable Remote wakeup */
          break;
          case USB_SETSTALL:
             /* Set stall */
          break;
          case USB_CLEARSTALL:
             /* Clear stall */
          break;
          default:
          break;
       }
    break;
    case  USB_CLASS:
       R_usb_pstd_ControlEnd(USB_DATA_ERR);
    break;
    case  USB_VENDOR:
       switch( data )
       {
          case USB_CS_IDST:            /* Idle or setup stage */
          case USB_CS_RDDS:            /* Control read data stage */
          case USB_CS_WRDS:            /* Control write data stage */
          case USB_CS_WRND:            /* Control write nodata status stage */
          case USB_CS_RDSS:            /* Control read status stage */
          case USB_CS_WRSS:            /* Control write status stage */
          case USB_CS_SQER:            /* Control sequence error */
          default:                     /* Illegal */
          break;
       }
       R_usb_pstd_SetStallPipe0();
    break;
    default:                                 /* Special function */
    break;
  }
}
```

*g_usb_LibPipe [pipe]->complete

Callback for data transfer end

Format

void (*g_usb_LibPipe[pipe]->complete)((usb_utr_t*)g_usb_LibPipe[pipe]);

Argument

g_usb_LibPipe Transfer message

Return Value

— —

Description

The data transfer end or forced is notified to the UPL.

Notes

1. A message when transfer is requested is returned. Table 6-7 shows the structure members updated by the USB-BASIC-F/W.
2. The PCD does not call back for the data transfer timeout (USB_DO_TRANSFER_TMO specified using the *R_usb_pstd_TransferEnd()* Function).

Table 6-7 usb_utr_t Structure Members

Members	Update	Function	Notes
tranlen	Updated	The remaining data length is notified. (tranlen = tranlen of the forwarding - data length actually sent or received)	
status	Updated	The following transfer results are notified. USB_DATA_OK When the data transfer (transmission / reception) normally ends. USB_DATA_SHT When the data transfer ends with less than the specified data length. USB_DATA_OVR When the received data size is exceeded USB_DATA_STOP When the data transfer is forcibly ended	
pipectr	Updated	The pipe control register (PIPExCTR register) value is notified	
Other than above	Not updated	The contents requested to be transferred are stored.	

Example

Processing of the function that callback is done as an example.

```
void usb_psmpl_transfer_result(usb_utr_t *mess)
{
  switch(mess->status)
  {
   case USB_DATA_OK:
   case USB_DATA_SHT:
      if (mess->keyword == USB_PIPE4)
      {
         usb_psmpl_DataTransfer(512, (uint8_t*)&g_usb_SmplTrnData);
      }
   break;
   case USB_DATA_OVR:
      if (mess->keyword == USB_PIPE5)
      {
         usb_psmpl_DataTransfer(512, (uint8_t*)&g_usb_SmplTrnData);
      }
   break;
  }
}
```

7. Host Sample Program (UPL)

This chapter assumes and explains the case where RL78 is used as MCU. The LowSpeed device cannot communicate the bulk transfer. Skip the description concerning the bulk transfer when the user system is LowSpeed device. R8C doesn't correspond to the LowSpeed device. Skip the description concerning the LowSpeed when the user system uses R8C as MCU.

The sample application performs the data communication when connected to the USB device.

7.1 Operating Environment

The Figure 7.1 and Figure 7.2 show a sample operating environment for the software.

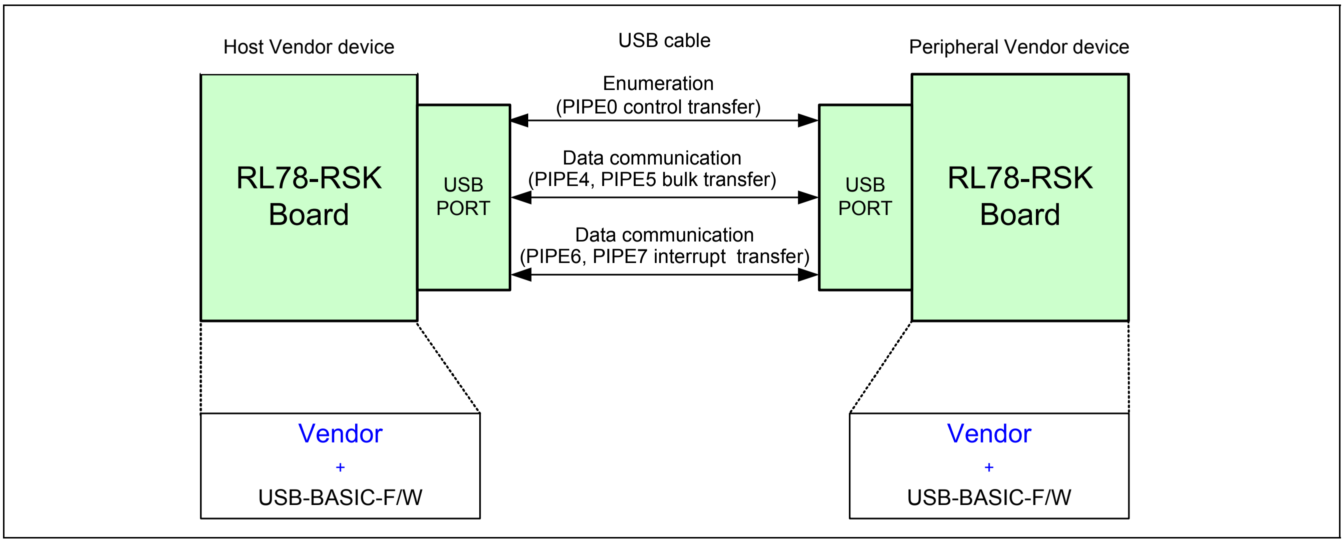

Figure 7.1 Example FullSpeed Operating Environment

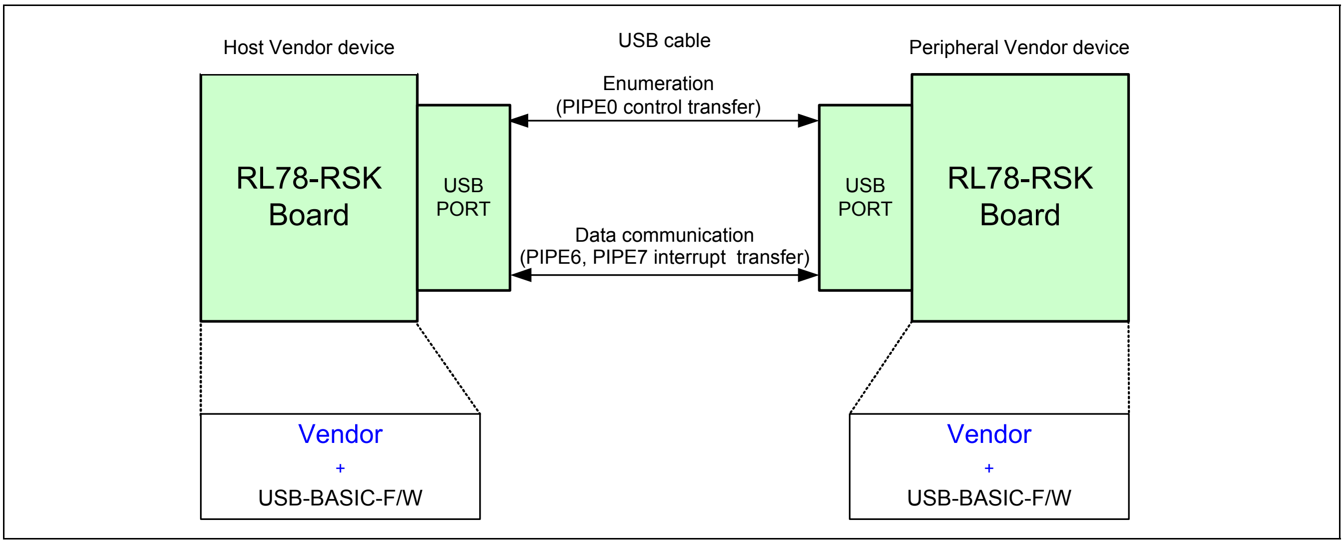

Figure 7.2 Example LowSpeed Operating Environment

7.2 Description of Host Sample Program

The host sample program of the USB-BASIC-F/W is operated with the FullSpeed device or the LowSpeed device that can be selected by the connected device. A sample program includes a vendor class driver and sample application for the data transfer. The data communication of bulk transfer uses the pipes 4 and 5. And the data communication of interrupt transfer uses the pipes 6 and 7. When creating a customer class driver or an application, refer to the *r_usb_vendor_hapl.c* file and *r_usb_vendor_hdriver.c* file. The following settings are necessary in order to communicate with a USB peripheral device as a USB host mode.

1. Setting a scheduler (the number of tasks, table size, task ID, and mail box ID, etc.)
2. Calling a task
3. Corresponding descriptor analysis processing to a device class driver to be mounted
4. Creating a corresponding pipe information table to a device class driver to be mounted.
5. Corresponding USB request forwarding to a device class driver to be mounted

7.2.1 Functions

Sample application

The USB state transition that is callback from the USB-BASIC-F/W is notified to the vendor class driver. And control the vendor class driver. At this time, when USB_STS_CONFIGURED is notified from the USB-BASIC-F/W, the global variable is initialized and the data transfer beginning is demanded to the vendor class driver. The data communications are bulk transfer using PIPE4 and 5, and interrupt transfer using PIPE6 and 7. When the end of data transfer is notified by the vendor class driver, the data transfer is restarted with the pipe that undertakes the notification. Moreover, the notification of the key input is received by regular processing. The sample of suspend, resume (in suspend state), and the port disable are included.

Vendor class driver

The global variable is initialized according to the USB state that is notified from APL. Moreover, when the data transfer is demanded from the application, the data transfer is demanded to USB-BASIC-F/W. The end of data transfer is notified to the application when the end of data transfer is notified from USB-BASIC-F/W. It doesn't correspond to the vendor class request.

Enumeration

When the USB host's connection is detected, USB-BASIC-F/W automatically starts enumeration. An enumeration ends normally, if the USB device that can work is connected. And USB_STS_CONFIGURED is notified to the application by the callback function.

Data communication

When an enumeration normally ends, the data transfer is possible. The application begins the data transfer of the USB state transition callback. It is possible to communicate with the device that operates USB-BASIC-F/W as the USB peripheral device.

Vendor class request

The vender class request is not issued.

USB state transition

To operates as follows by the notification of the USB state transition.

USB_STS_DETACH:	Stop the data transfer, Initialized application sequence mode
USB_STS_DEFAULT:	No processing
USB_STS_ADDRESS:	No processing
USB_STS_CONFIGURED:	Pipe registration, Start the data transfer
USB_STS_SUSPEND:	No processing
USB_STS_RESUME:	No processing
USB_STS_WAKEUP:	The same as the resume processing

It is possible to return from the state of the suspended by the remote wakeup signal. Moreover, it is also possible to demand suspend and resume from the application to USB-BASIC-F/W.

Driver check callback

When the Configuration descriptor is acquired in the sequence processing of enumeration, USB-BASIC-F/W executes driver confirmation callback function (**g_usb_hstd_Driver.classcheck*) registered in USB-BASIC-F/W. The application confirms operation right or wrong of connected device to the vendor class driver by the *R_usb_hvndr_ClassCheck()* function. The item to which the vendor class driver judges right or wrong of operation is as follows:

1) Are notified VID and PID corresponding to the vendor driver ?
2) Is there a string descriptor of the product ID ?
3) Are two Bulk pipes and two Interrupt pipes in the interface?

The vendor driver responds to the USB-BASIC-F/W he answer of USB_YES by API function *R_usb_hstd_ReturnEnuMGR()* when all requirements are met.

7.2.2 Operation of Host Sample Program

1. Initial setting
   - For HEW

   When performing hardware reset for a device, the *_PowerON_Reset_PC* function in *ncrt0.a30* is called. The reset function initializes the MCU and call the hardware initialization function *usb_cpu_mcu_initialize()* function. When returning from the hardware initialization function, initialize memory areas and calls the *main()* function in *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

   - For CubeSuite+

   When performing hardware reset for a device, the *_@cstart* function of a startup file created using the CubeSuite+ is called. The startup function initializes the MCU, and call the hardware initialization function *hdwinit()* function of the user definition. When returning from the hardware initialization function, initialize memory areas such as *sadd*r area and call the *main()* function in the *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

   - For EWRL78

   When performing hardware reset for a device, the *_@cstart* function in cstartup.s87 is called. The startup function initializes the MCU, and call the hardware initialization function *hdwinit()* function of the user definition. When returning from the hardware initialization function, initialize memory areas such as *sadd*r area and call the *main()* function in the *main.c* file. For more details of startup processing, refer to HM and integrated development environment manual.

2. main function processing

   The *main()* function initializes the system by the *usb_hsmpl_main_init()* function(initialization of target MCU and the board, initialization of the USB module, starts of USB-BASIC-F/W, registration of the UPL driver's, and operation permission of the USB module), and the program is in the static state that wait for a request generation in the main loop.
   Operations of the main loop are as follows:
   (1) Determine a request in a scheduler.
   (2) When processing is requested, start a task.
   (3) Perform static processing.
   (4) Return to (1).

3. Sample application task (*usb_hsmpl_apl_task()*)

   When an enumeration normally ends, the sample application initializes global variables and requests the start of data transfer using API function *R_usb_hvndr_TransferStart()* to the vendor class driver. When a transfer end callback is received from the vendor class driver, the data transfer is repeated using API function *R_usb_hvndr_TransferStart()*.

4. Vendor class driver (*R_usb_hsmpl_VendorTask()*)

   When the data transfer demand is notified from the sample application, the vendor class driver (HDCD) demands the data transfer to USB-BASIC-F/W using API function *R_usb_hstd_TransferStart()*. Moreover, the end of data transfer is notified to the application by the callback function when the forwarding end callback is received from USB-BASIC-F/W.
   Even if the USB state transition is notified from the sample application to the vender class driver, special processing is not done. It starts / ends of the vendor class driver, the application sets the register of pipe information judging the USB state, and the data transfer begins.

Figure 7.3 shows the outline flow of the UPL.

The USB-BASIC-F/W comprises tasks that implement control functions for USB data transmit and receive operations. When an interrupt occurs, a notification is sent by means of a message to the USB-BASIC-F/W. When the USB-BASIC-F/W receives a message from the USB interrupt handler, it determines the interrupt source and executes the appropriate processing.

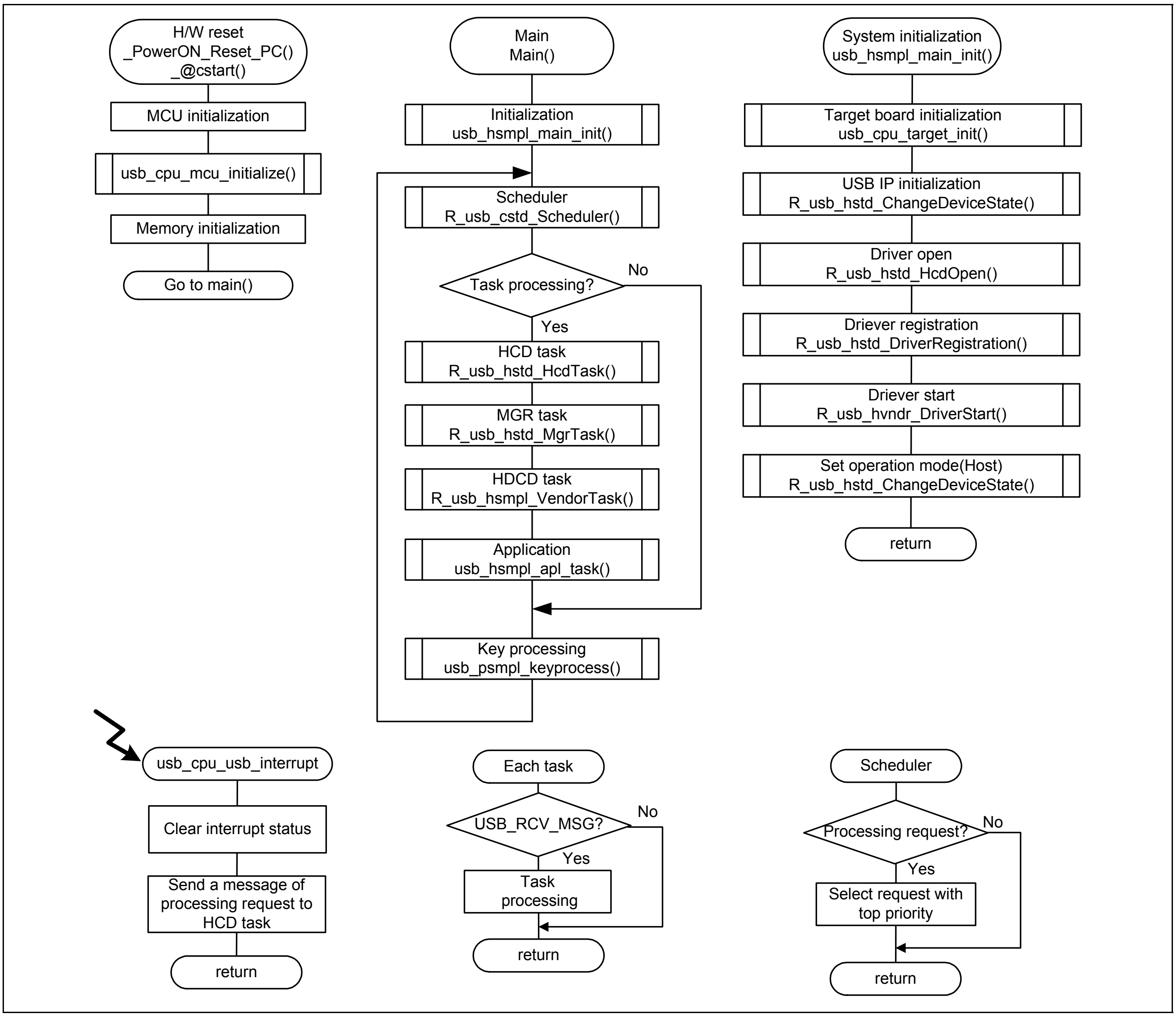

Figure 7.3 Sequence Outline

7.2.3 Setting a Scheduler

Set the maximum value of a task ID, and maximum value of a message stored in the task priority table at *r_usb_cstd_kernelid.h* file.

```
/* Please set with user system */
#define    USB_IDMAX       ((uint8_t)5)    /* Maximum Task ID +1 */
#define    USB_TABLEMAX    ((uint8_t)5)    /* Maximum priority table */
#define    USB_BLKMAX      ((uint8_t)5)    /* Maximum block */
```

7.2.4 Setting a Task ID and Mail Box ID

Set a task ID and mail box ID at *r_usb_cstd_kernelid.h* file.
The task priority level is the same as task ID. (When the task identification number is small, priority is high.)

```
#define    USB_HCD_TSK     USB_TID_0       /* Host Control Driver Task */
#define    USB_HCD_MBX     USB_HCD_TSK     /* Mailbox ID */
#define    USB_MGR_TSK     USB_TID_1       /* Host Manager Task */
#define    USB_MGR_MBX     USB_MGR_TSK     /* Mailbox ID */
#define    USB_HVEN_TSK    USB_TID_2       /* Task ID */
#define    USB_HVEN_MBX    USB_HVEN_TSK    /* Mailbox ID */
#define    USB_HSMP_TSK    USB_TID_3       /* Host Sample Task */
#define    USB_HSMP_MBX    USB_HSMP_TSK    /* Mailbox ID */
```

7.2.5 Task calling

Call a task to be used in main loop (*main()* function).

```
void main (void)
{
  usb_hsmpl_main_init();

  /* Sample main loop  */
  while( 1 )
  {
    if( R_usb_cstd_Scheduler() == USB_FLGSET )
    {
       R_usb_hstd_HcdTask();    /* HCD Task */
       R_usb_hstd_MgrTask();    /* MGR Task */
       R_usb_hsmpl_VendorTask();
       usb_hsmpl_apl_task();
    }
  }
}
```

7.2.6 Starting the UPL

When the USB-BASIC-F/W establishes a structure with a device (SET_CONFIGURATION request issued), a device connection is notified to the UPL that can be operated using the callback function (**g_usb_HcdDriver.statediagram*). Analyze the USB sate of the second argument and perform the suited processing to a system. The sample application initializes the data area, puts the pipe configuration register to enabled state and begins the data transfer.

7.2.7 Application Outline

The USB-BASIC-F/W starts the data transfer after configuration in the procedure shown below.
Identify the USB state using callback function *usb_hsmpl_device_state()* and request to vendor class driver the data transfer.

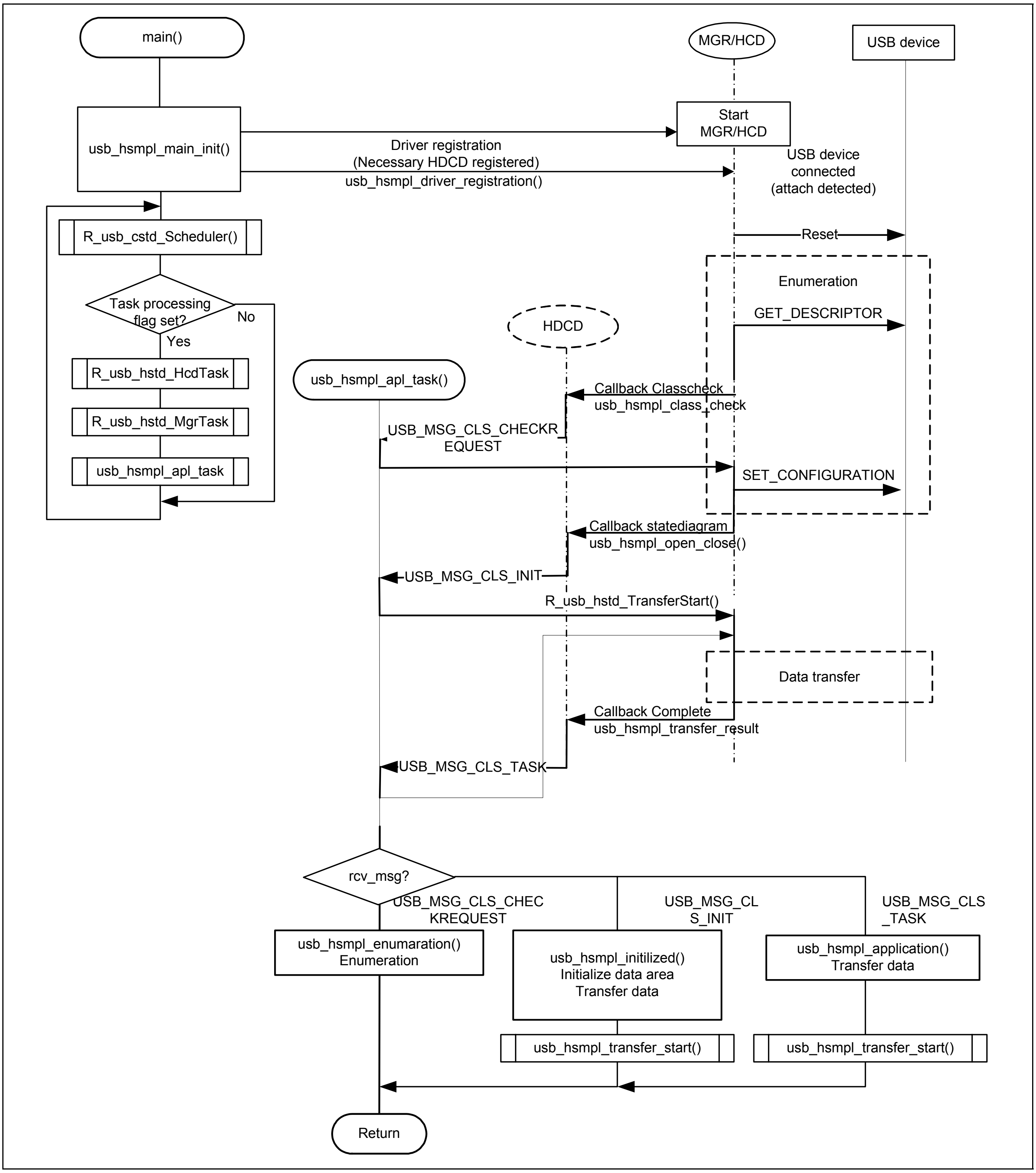

Figure 7.4 Application Operation Outline

7.3 Data Transfer and Control Transfer

The data transfer is the customer-specific function specification and a transfer method, communication start or end timing, and buffer structure need to be changed based on a system.

7.3.1 Basic specification

As for the USB-BASIC-F/W, the data transfer is possible using the user buffer specified by *usb_utr_t* structure in Table 8-4. When the data transfer ends, the USB-BASIC-F/W sets *PID = NAK* and notifies the transfer end by the callback function.

The USB-BASIC-F/W updates the pipe status (data toggle) to pipe status (*utr_table.pipectr*) specified when the data transfer is demanded. Moreover, the pipe status (data toggle) is notified by the callback of the data transfer end. Therefore, because UPL memorizes the pipe status, the data transfer of "the control is exclusively possible" plural endpoint to which the data transfer is not done at the same time with one pipe is possible. The pipe status however should initialize at “*DATA0*" by factors of USB reset, the STALL release, the SET_CONFIGURATION request, and the SET_INTERFACE request, etc.

The size of the max packet of the Bulk pipe is a setting of 64 byte fixation. When the host operations, the max packet size of the default pipe immediately after the issue of USB reset doesn't do the error judgment.

7.3.2 Data Transfer Request

Use *R_usb_hstd_TransferStart()* to start the data transfer.

7.3.3 Control Transfer Request

Use *R_usb_hstd_TransferStart()* to start the data transfer. Please refer to Table 8-3 for the specification of the setup packet. The control transfer is not done when there is an error in the setup packet.

7.3.4 Notification of Transfer Result

The data transfer end is notified to the UPL using the callback function specified in the *usb_utr_t* structure. Refer to Table 8-8 for the content to be notified.

7.3.5 Notes on Data Reception

(1) Use a transaction counter for the received pipe.

When a short packet is received, the expected remaining receive data length is stored in tranlen of *usb_utr_t* structure and this transfer ends. When the received data exceeds the buffer size, data read from the FIFO buffer up to the buffer size and this transfer ends. When the user buffer area is insufficient to accommodate the transfer size, the *usb_cstd_forced_termination()* function may clear the receive packet.

(2) Received callback

When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated. When receiving a short packet or the data size is satisfied, the USB-BASIC-F/W judges the transfer end and generates the callback.

Example

When the data size of the reception schedule is 128 bytes and the maximum packet size is 64 bytes:

1 to 63 bytes received	A received callback is generated.
64 bytes received	A receive callback is not generated.
65 to 128 bytes received	A receive callback is generated.

7.3.6 Data transfer Outline

Set the necessary information to the *usb_uttr_t* structure and call *R_usb_hstd_TransferStart()*. Examples of the control transfer and data transfer are shown as follows.

Example of data transfer

```
void usb_hsmpl_transfer_start( uint16_t pipe )
{
  if( g_usb_SmplTrnCnt[pipe] != 0 )
  {
   g_usb_SmplTrnMsg[pipe].keyword   = pipe;     /* Data area address */
   g_usb_SmplTrnMsg[pipe].tranadr   = g_usb_SmplTrnPtr[pipe];
   g_usb_SmplTrnMsg[pipe].tranlen   = g_usb_SmplTrnSize[pipe];
   g_usb_SmplTrnMsg[pipe].setup     = (uint16_t*)USB_NULL;
   g_usb_SmplTrnMsg[pipe].complete  = (usb_cb_t)&usb_hsmpl_transfer_result;
   R_usb_hstd_TransferStart((usb_utr_t*)&g_usb_SmplTrnMsg[pipe]);
  }
}
```

Example of control transfer

```
usb_er_t usb_hstd_set_configuration(void)
{
  g_usb_MgrRequest.WORD.BYTE.bmRequestType =
      USB_REQUEST_TYPE(USB_HOST_TO_DEV,USB_STANDARD,USB_DEVICE);
  g_usb_MgrRequest.WORD.BYTE.bRequest      = USB_SET_CONFIGURATION;
  g_usb_MgrRequest.wValue                  =
     (uint16_t)(g_usb_MgrConfDescr[USB_CON_CONFIG_VAL]);
  g_usb_MgrRequest.wIndex                  = 0x0000;
  g_usb_MgrRequest.wLength                 = 0x0000;
  g_usb_MgrRequest.Address                 = (uint16_t)g_usb_MgrDevAddr;

  g_usb_MgrControlMessage.tranadr    = (void*)data_table;
  g_usb_MgrControlMessage.complete   = (usb_cb_t)&usb_hstd_transfer_result;
  g_usb_MgrControlMessage.tranlen    = (usb_leng_t)g_usb_MgrRequest.wLength;
  g_usb_MgrControlMessage.pipenum    = USB_PIPE0;
  g_usb_MgrControlMessage.setup      = (void*)&g_usb_MgrRequest;
  R_usb_hstd_TransferStart(&g_usb_MgrControlMessage);
}
```

Example of the callback function (notify the transfer end to the UPL task via a message) is shown as follows.

```
void usb_hsmpl_transfer_result(usb_utr_t *mess)
{
  mess->msginfo = USB_MSG_CLS_TASK;         /* Data transfer Callback */
  USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)mess);
}
void usb_hstd_transfer_result(usb_utr_t *mess)
{
  g_usb_MgrSequence++;
  utrmsg->msginfo  = USB_MGR_CONTINUE;        /* Enumeration */
  USB_SND_MSG(USB_MGR_MBX, (usb_msg_t*)mess);
}
```

7.4 Pipe Information

The pipe setting suited to this class driver needs to be retained as the pipe information table. The pipe information of a device is described in *uint16_t g_usb_hvndr_DefEpTbl[]* of the *r_usb_vendor_hdriver.c* file by the vendor class driver.

7.4.1 Pipe Information Table

A pipe information table comprises the following four items (uint16_t × 4).

1. Pipe window select register (address 0x64)
2. Pipe configuration register (address 0x68)
3. Pipe maximum packet size register (address 0x6C)
4. Pipe interval register (address 0x6E)

7.4.2 Pipe Definition

The pipe information table structure used in the vendor class driver is shown below. The macros are defined in the *r_usb_cstd_defusbip.h* file and refer to the header file by the pipe definition item of the pipe information table for the assignable value.

Structure example of pipe information table:

```
uint16_t g_usb_hvnr_DefEpTbl[] =                              ← Pipe information table
{
    USB_PIPE4,                                                ← Pipe definition item 1
    USB_NULL|USB_BFREOFF|USB_DBLBOFF|USB_SHTNAKOFF,           ← Pipe definition item 2
    USB_NULL,                                                 ← Pipe definition item 3
    USB_NULL,                                                 ← Pipe definition item 4
            :
    USB_PDTBLEND,                                             ← Pipe information table end definition
}
```

(1) Pipe definition item 1: Specify the value set to the pipe window select register
Pipe selected: Specify pipes to be selected (USB_PIPE4 to USB_PIPE7)

(2) Pipe definition item 2: Specify the setting value of the pipe configuration register.

Transfer Type	:	Specify either USB_BULK or USB_INT.
BRDY interrupt operation specified	:	Specify USB_BFREOFF
Double buffer mode	:	Specify either USB_DBLBON or USB_DBLBOFF
SHTNAK operation specified	:	Specify either USB_SHTNAKON or USB_SHTNAKOFF
Transfer direction	:	Specify USB_DIR_H_OUT or USB_DIR_H_IN
Endpoint number	:	Specify endpoint number (EP1 to EP15) to pipes

- The settable values differ depending on the selected pipes for the transfer type. For details, refer to the User's Manual: Hardware.
- Describe the pipe information according to the endpoint descriptor of connecting device.
- Set USB_SHTNAKON for the receive direction pipe (USB_DIR_H_IN).

(3) Pipe definition item 3: Specify the device address and the maximum packet size of the endpoint.
- Specify the device address: Set the device address by using the *USB_ADDR2DEVSEL* macro.
- Specify the maximum packet size: Set the value based on the USB specification.

(4) Pipe definition item 4: Specify the interval time of the endpoint.
- Interval time specified: Set the value according to the User's Manual: Hardware.

(5) Others.

- The pipe information is necessary for the number of endpoints that can be communicated simultaneously.
- Synchronize communication each transfer in the UPL.
- Please manage the pipe information used with the UPL.
- Write USB_PDTBLEND at the end of the table.
- The USB-BASIC-F/W notifies the device state transition by the callback function, to mount the register setting (release) processing of the pipe information by using API function on the UPL side.

API function *R_usb_hstd_ChkPipeInfo()* that sets the transfer type, transfer direction, endpoint number, maximum packet size, and interval time from the endpoint descriptor is provided. When using this function, specify "USB_NULL" for the each field.

7.5 In Order to Operate the USB-BASIC-F/W by the Host mode

This chapter describes a procedure to operate the USB-BASIC-F/W by the host mode as an example of the sample code.

7.5.1 Select a device

Table 7-1 lists the integrated development environment of each device provided by the USB-BASIC-F/W and hardware resources. Use a corresponding folder to a device to be used in the Project folder for each device. Please change the H/W resource folder name of the board used from "*HwResourceForUSB_G1C _ **board name***" to "*HwResourceForUSB_G1C*" when MCU is RL78.

Table 7-1 Hardware Resource of Sample Code

<table>
<tr><th>Device</th><th>Integrated development environment</th><th>Host</th><th>Data rate</th><th>Hardware Resource Folder</th></tr>
<tr><td rowspan="4">R8C/3MU,
R8C/34U,
R8C/3MK,
R8C/34K</td><td rowspan="4">HEW</td><td>—</td><td>FullSpeed</td><td rowspan="4">R8C\HwResourceForUSB</td></tr>
<tr><td>—</td><td>FullSpeed</td></tr>
<tr><td>1PortHost</td><td>FullSpeed</td></tr>
<tr><td>1PortHost</td><td>FullSpeed</td></tr>
<tr><td rowspan="2">RL78/G1C</td><td rowspan="2">CubeSuite+/
EWRL78</td><td>1PortHost</td><td>FullSpeed
LowSpeed</td><td>RL78\HwResourceForUSB_G1C_RSK*1</td></tr>
<tr><td>2PortHost</td><td>FullSpeed
LowSpeed</td><td>RL78\HwResourceForUSB_G1C_EVA</td></tr>
</table>

Note)

*1: The connector for the USB host mode of RL78-RSK is connected with the *USB-PORT1* side. The USB-BASIC-F/W doesn't support one port host mode only on the *USB-PORT1* side. Therefore, the execution file works as one port host by making as two port host (USB_PORTSEL_PP=USB_2PORT_PP), and using the *USB-PORT1* side.

7.5.2 User Definition Information

Rewrite the user definition information file (*r_usb_cstd_defusr.h*) in the "*inc*" folder to set the function of the USB-BASIC-F/W. Settable items and outline in the user definition information file are shown below.

(1) Specify an operating mode

Set an operating mode of the USB module.
#define USB_FUNCSEL_PP USB_HOST_PP : Operate USB in a host mode

(2) Specify the USB port

Set the number of USB ports to be used (this item will be used only for the RL78)
#define USB_PORTSEL_PP USB_1PORT_PP : Use a USB port
#define USB_PORTSEL_PP USB_2PORT_PP : Use two USB ports

(3) Specify the battery charging operation (Only RL78/USB)

Set the battery charging operation. Make the macro in operation effective.
#define USB_HOST_BC_ENABLE : Enable batetry charging

Set the dedicated charging port operation. Make the macro in operation effective.
#define USB_BC_DCP_ENABLE : Dedicated Charging Port

The operation mode of USB-BASIC-F/W is specified with not the header file but the project file of the integration environment (build configuration of HEW or build mode of CubeSuite+ or Configurations of EWRL78).

A set content can be confirmed in the project file of CubeSuite+ according to the following procedures.

(1) Double click "*\Project\RL78\RL78G1C_48pin.mtpj*" to start the CubeSuite+.
(2) Open a property in CA78K0R (build tool) of a project tree.
(3) Click a table of a compile option.
(4) Items of the definition macro are defined as follows:

Definition macro	definition macro [3]
[0]	USB_FUNCSEL_PP=USB_HOST_PP
[1]	USB_PORTSEL_PP=USB_2PORT_PP
[2]	RL78USB

A set content can be confirmed in the project file of HEW according to the following procedures.

(1) Double click "*\Project\R8C\R8C34K.hws*" to start the HEW.
(2) Select build to open "*Renesas M16C Standard Toolchain*"
(3) Open "*category: source*" and "*option: identifier definition*".
(4) Items of identifier definition are defined as follows:

Define	Value
USB_FUNCSEL_PP	USB_HOST_PP
USB_PORTSEL_PP	USB_1PORT_PP
R8CUSB	

A set content can be confirmed in the project file of EWRL78 according to the following procedures.

(1) Double click "*RL78G1C_48pin.eww*" to start the EWRL78.
(2) Click "*Project*" and "*Options*".
(3) Click a table of Preprocessor in C/C+compiler category.
(4) Items of the defined symbol are defined as follows:

USB_FUNCSEL_PP=USB_HOST_PP
USB_PORTSEL_PP=USB_2PORT_PP
RL78USB

7.5.3 Setting Example of Host Operation

Setting procedure to operate the USB-BASIC-F/W as the host mode is shown below using the CubeSuite+ as an example.

<Setting procedure>

1. Select a hardware resource
   Change the folder name of "*HwResourceForUSB_RL78_G1C_* ***board name***" to "*HwResourceForUSB_G1C*".
2. Set a workspace
   Double click "*\Project\RL78\RL78G1C_48pin.mtpj*" and to start the CubeSuite+. Then select "*Host_1Port-RSK*" in build mode of the workspace.
3. Create an execute file
   Select "*build*" and "*rebuild project*" from the upper tab of the workspace to build.
4. Set the debug environment
   Select a debug tool to use in a debug tool of a project.
5. Connect with the evaluation board
   Select "*debug*" and "*connect*" from the upper tab of the workspace to start the debug environment.
6. Download and execute an execute file
   Select "*debug*" and "*download to debug tool*" from the upper tab of the workspace to download the execute file to the debug environment.
7. Execute an execute file
   Select "*debug*" and "*execute after reset*" from the upper tab of the workspace to execute a program.

Setting procedure to operate the USB-BASIC-F/W as the host mode is shown below using the HEW as an example.

< Setting procedure >

1. Set a workspace
   Double click "*Fw.hws*" to start the HEW. Then select "*HOST*" in the configuration of the workspace.
2. Create an execute file
   Select "*build*" and "*rebuild project*" from the upper tab of the workspace to build.
3. Set the debug environment
   Select a debug tool to use in a debug tool of a project.
4. Connect with the evaluation board
   Select "*debug*" and "*connect*" from the upper tab of the workspace to start the debug environment.
5. Download an execute file
   Select "*debug*" and "*download to debug tool*" from the upper tab of the workspace to download the execute file to the debug environment.
6. Execute an execute file
   Select "*debug*" and "*execute after reset*" from the upper tab of the workspace to execute a program.

Setting procedure to operate the USB-BASIC-F/W as the peripheral mode is shown below using the EWRL78 as an example.

<Setting procedure>

1. Select a hardware resource
   Change the folder name of "*HwResourceForUSB_RL78_G1C_* ***board name***" to "*HwResourceForUSB_G1C*".
2. Set a workspace
   Double click "*RL78G1C_48pin.eww*" and to start the EWRL78. Then select "*Host_1Port-RSK*" from the upper pull down menu of the workspace.
3. Create an execute file

Select "*Project*" and "*Make*" from the upper tab of the workspace to build.

4. Set the debug environment
   Select "*Project*" and "*Options*". Then select a debug tool to use at Setup tab in Debugger category.
5. Connect with the evaluation board and Download an execute file
   Select "*Project*" and "*Download and Debug*" from the upper tab of the workspace to download the execute file to the debug environment.
6. Execute an execute file
   Select "*Debug*" and "*Go*" from the upper tab of the workspace to execute a program.

7.5.4 Change the USB-BASIC-F/W

The program and header file shown below need to be changed in order to operate the USB-BASIC-F/W.
Sample functions for the Renesas USB MCU will be provided. Change them according to the user system.

- Initializes the MCU for control, interrupt handler, interrupt control, and etc. Refer to Table 7-2.
- Time adjustment of specified time wait function (*usb_cpu_delay_xms()* function, *usb_cpu_delay_1u()* function)
  Generate the specified wait time using loop processing. Adjust for the time in order to generate the specified time, such as changing the number of loops according to the system.
- Set the function to disable and enable the USB associated interrupts in order to use the scheduler function. (*usb_cpu_int_disable()* function, *usb_cpu_int_enable()* function)
  The USB interrupt disable function (*usb_cpu_int_disable()* function) disables the USB interrupt and the USB interrupt enable function (*usb_cpu_int_enable()* function) enables the USB interrupt for the USB-BASIC-F/W. Perform the setting according to the MCU to be used.

Table 7-2 Function List

Type	Function Name and argument	Description	Notes
void	usb_cpu_mcu_initialize(void)	MCU initialization (oscillation control, etc.)	
void	usb_cpu_target_init(void)	System initialization	
void	usb_cpu_set_pin_function (uint16_t function)	USB function setting of the MCU(pin setting, etc.)	
void	usb_cpu_usb_interrupt (void)	USB interrupt handler	
void	usb_cpu_usbint_init (void)	USB interrupt enabled	
void	usb_cpu_int_enable(void)	USB interrupt enabled for the scheduler	
void	usb_cpu_int_disable(void)	USB interrupt disabled for the scheduler	
void	usb_cpu_delay_1us(uint16_t time)	1 μs wait processing	
void	usb_cpu_delay_xms(uint16_t time)	1 ms wait processing	
void	usb_cpu_stop_mode(void)	Execute the STOP instruction	

7.5.5 User System Definition Information File (*r_usb_usrconfig.h*)

(1) Control read data buffer size

Specify the data buffer size received in the control read transfer.
Example: Device descriptor 20 bytes, configuration descriptor 256 bytes

```
#define     USB_DEVICESIZE      20u
#define     USB_CONFIGSIZE      256u
```

(2) Device address

Specify the device address connected to PORT0.
Example: When starting a device address from 2

```
#define     USB_DEVICEADDR      2u
```

Addresses can be specified from 1 to 5. However, specify the address within the range of 1 to 4 when you use the *USB-PORT1* side.

(3) Debounce interval

Specify the debounce interval time after attach.
Example: Until the scheduler is passed 3000 times

```
#define     USB_TATTDB          3000
```

The debounce interval is a minimum duration of 100ms to provide by the USB System Software (See USB specification Chapter 7.1.7.3). Only the predetermined number passes the main loop, the USB-BASIC-F/W outputs the USB reset signal to the connected device.

8. Host Control Driver (HCD)

8.1 Basic Information

The HCD is a program to control the hardware when operating the target device as the USB host mode. The USB-BASIC-F/W analyzes generation issued from the UPL and controls the H/W. The H/W control result is notified to the UPL using the return value of the API function and callback function. A request from the H/W is notified using the callback function of the driver information registered in the USB-BASIC-F/W. Start the USB-BASIC-F/W shown in Chapter 8.2.1 and register the UPL shown in Chapter 8.2.2 to make the USB-BASIC-F/W as a host device.

USB-BASIC-F/W functions are shown below.

1. Detection for the USB state change with the connected device and notification for the change result: Chapter 8.2.3
2. Enumeration with the connected device: Chapter 8.2.8
3. Operation of the connected device can be determined: Chapter 8.2.4
4. Data transfer and transfer result notification: Chapter 8.2.5
5. USB state control (USB state control and notification for control result): Chapter 8.2.7

8.2 Operation Outline

8.2.1 Starting the HCD

Start the USB-BASIC-F/W using API function *R_usb_hstd_HcdOpen()*.

8.2.2 Registration for the UPL

The UPL registers the information in Table 8-1 below to the USB-BASIC-F/W using API function *R_usb_hstd_DriverRegistration()*.

USB-BASIC-F/W preserves information in global variable (*g_usb_HcdDriver[]*).

```
typedef struct
{
  usb_port_t      rootport;      /* Root port */
  usb_addr_t      devaddr;       /* Device address */
  uint16_t        devstate;      /* Device state */
  uint16_t        ifclass;       /* Interface Class */
  usb_cb_check_t  classcheck;    /* Driver check */
  usb_cb_info_t   statediagram;  /* Device status */
} usb_hcdreg_t;
```

Table 8-1 Member of usb_hcdreg_t Structure

Members	Functions	Notes
rootport	USB-BASIC-F/W uses this variable. The connected port number is registered.	
devaddr	USB-BASIC-F/W uses this variable. The device address is registered.	
devstate	USB-BASIC-F/W uses this variable. The device connection state is registered.	
ifclass	Register the interface class code in which the UPL operates.	
classcheck	Register a function to check the connecting device operation for the enumeration.	
statediagram	Register a function to start when the USB state is transited.	

8.2.3 Notification for USB State Change

To notify UPL the USB state transition etc, the USB-BASIC-F/W executes USB state transition callback function (**g_usb_PcdDriver.statediagram*) for UPL registered in USB-BASIC-F/W. The USB-BASIC-F/W notifies the information below to the UPL using the second argument of the callback function. Analyze the USB sate and perform suitable processing to the system.

USB state transition

USB_STS_DETACH:	Detach detection
USB_STS_ATTACH:	Attach detection
USB_STS_DEFAULT:	Default state transition (USB reset detection)
USB_STS_OVCRCURRENT:	Over current detection
USB_STS_CONFIGURED:	Configured state transition (Set_Configuration request transmission)
USB_STS_WAKEUP:	Configured state transition (remote wakeup processing ends)
USB_STS_POWER:	Enable a port (request using the API function)
USB_STS_PORTOFF:	Disable a port (request using the API function)
USB_STS_SUSPEND:	Suspend (request using the API function)
USB_STS_RESUME:	Resume (request using the API function)
USB_STALL_SUCCESS:	Cancel STALL for the peripheral device (request using the API function)

8.2.4 Operation right or wrong judgment of connected device

When the USB-BASIC-F/W detects the device connect, enumeration shown in Chapter 8.2.8 is performed. When the Configuration descriptor is obtained in the sequence processing of the enumeration and the driver check callback function (**g_usb_hstd_Driver.classcheck*) that registered in USB-BASIC-F/W is executed. The USB-BASIC-F/W notifies the information in Table 8-2 below to the UPL in the first argument of the callback function. To be analyze the received device information by the UPL. Obtain if the information other than descriptions listed in Table 8-2 is necessary using the API function *R_usb_hstd_TransferStart()*. When identifying the connected device, return operation (USB_YES/USB_NO) to the USB-BASIC-F/W using the API function *R_usb_hstd_ReturnEnuMGR()*. When USB_YES is notified, the USB-BASIC-F/W continues the enumeration and transits the device to configured state. When USB_NO is notified, registration for other operable drivers is searched.

```
table[0] = (uint16_t*)&g_usb_MgrDeviceDescriptor;
table[1] = (uint16_t*)&g_usb_MgrConfigurationDescriptor;
table[2] = (uint16_t*)&g_usb_HcdDeviceAddr;
(*driver->classcheck)((uint16_t**)&table);
```

Table 8-2 Argument Array of classcheck

Order of Array	Functions	Notes
table[0]	Address of device descriptor storage area	
table[1]	Address of configuration descriptor storage area	
table[2]	Address of global variable that mean the Device Address	

8.2.5 Data transfer Request and Notification to the USB-BASIC-F/W

Please assume the following structures to be an argument and call an API function *R_usb_hstd_TransferStart()*, when the UPL wants to the data transfer. The USB-BASIC-F/W preserves address information of the argument in global variable (*g_usb_LibPipe*). Therefore, please maintain the realities of the argument in UPL until the data transfer ends.

```
struct usb_utr_t
{
  usb_strct_t   msginfo;       /* Message Info for F/W */
  usb_strct_t   pipenum;       /* Pipe number */
  usb_strct_t   status;        /* Transfer status */
  usb_strct_t   flag;          /* Flag */
  usb_cb_t      complete;      /* Call Back Function Info */
  void          *tranadr;      /* Transfer data Start address */
  uint16_t      *setup;        /* Setup packet(for control only) */
  uint16_t      pipectr;       /* Pipe control register */
  usb_leng_t    tranlen;       /* Transfer data length */
  uint8_t       dummy;         /* Adjustment of the byte border */
}
```

8.2.6 Setup Packet

Set the address of the following structures to member **setup* of the *usb_utr_t* when the control transfer is executing.

```
typedef struct
{
  union {
   struct {                       /* Characteristics of request */
      uint8_t   bmRequestType;    /* Characteristics of request */
      uint8_t   bRequest;         /* Specific request */
   } BYTE;
   uint16_t wRequest;             /* Control transfer request */
  } WORD;
  uint16_t      wValue;           /* Control transfer value */
  uint16_t      wIndex;           /* Control transfer index */
  uint16_t      wLength;          /* Control transfer length */
  uint16_t      Address;
} usb_hcdrequest_t;
```

Table 8-3 Member of usb_hcdrequest_t Structure

Members	Functions	Notes
bmRequestType	The value is bmRequestType of request. Set this member by using the *USB_REQUEST_TYPE* macro.	
bRequest	The value is bRequest of request.	
wRequest	The value is wRequest of request. (The value is BREQUEST of USBREQ register.) The bit can refer for wRequest in a union type.	
wValue	The value is wValue of request. (The value is USBVAL register.)	
wIndex	The value is wIndex of request. (The value is USBINDEX register.)	
wLength	The value is wLength of request. (The value is USBLENG register.)	
Address	Device address	

Table 8-4 Member of usb_utr_t Structure

Members	Functions	Notes
msginfo	This member is message information that USB-BASIC-F/W uses. It is set using the API function.	
pipenum	Specify the pipe number in the UPL.	
status	The USB-BASIC-F/W returns the following status information. USB_CTRL_END: Control transfer normal end USB_DATA_OK: Data transfer (transmission/reception) normal end USB_DATA_SHT: Data reception normal end with less than specified data length USB_DATA_OVR: Receive data size exceeded USB_DATA_ERR: No-response condition or over/under run error detected USB_DATA_DTCH： Detach detected USB_DATA_STALL: STALL or Max packet size error detected USB_DATA_STOP: Data transfer forced end USB_DATA_TMO: Forced end due to timeout, no callback	
flag	Not used	
complete	Specify the callback function to be executed in the UPL at the data transfer end. Type declaration is as follows: typedef void (*usb_cb_t)(usb_utr_t*);	
*tranadr	Specify the following information for the UPL. Reception or ControlRead: Buffer address to store the receive data Transmission or ControlWrite: Buffer address to store the transmit data NoDataControl transfer: Ignored if specified To secure the bigger area than the data length at the specified with tranlen.	
*setup	When performing the control transfer, specify the structure address of Table 8-3.	
pipectr	Specify the PIPExCTR register information for the UPL. Control the sequence bit of DATA0/DATA1 according to bit 6 of the applicable member. Set USB_NULL for the initial state and the returned value by the USB-BASIC-F/W after the second communication. USB-BASIC-F/W returns the PIPECTR register information.	
tranlen	Specify the following information for the UPL. Reception or ControlRead transfer: Data length to be received Transmission or ControlWrite transfer: Data length to be transmitted NoDataControl transfer: Specify 0. The remaining transmit/receive data length is stored for the HCD after USB communication end. The maximum length that can be sent and received is 65535 bytes. USB-BASIC-F/W stored the remaining transmit/receive data length after the end of data transfer. The tranlen is remaining data length of the data stage, when a control transfer occurs.	

8.2.7 Request and Notify to Change the USB State for the HCD

To call API function *R_usb_hstd_MgrChangeDeviceState()*, when the UPL wants to change the USB state. Indicate the controlled descriptions using the API function argument. A UPL request is notified message to the USB-BASIC-F/W, the MGR task executes the state transition while controlling the sequence. When the USB state change of the connected device ends, the result is notified using the callback function. Information controlled by the USB-BASIC-F/W can be obtained using the API function *R_usb_hstd_DeviceInformation()*.

8.2.8 Enumeration

When a USB device connection is detected from the USB-BASIC-F/W, issue the USB reset and perform enumeration. In sequence processing of enumeration, issue the standard request below. The USB-BASIC-F/W allocates the address of “USB_DEVICEADDR” defined by a user macro to the device connected to port 0. When the H/W supports port 1, allocate the address of “USB_DEVICEADDR+1” to the device connected to port 1. However, please define the macro of "USB_DEVICEADDR" so that the address number should not exceed "0x05".

(1) GET_DESCRIPTOR (Device Descriptor)

(2) SET_ADDRESS

(3) GET_DESCRIPTOR (Configuration Descriptor)

(4) SET_CONFIGURATION

After the configuration descriptor is obtained, callback function (See Chapter 8.2.4) registered in USB-BASIC-F/W is executed. The UPL confirms right or wrong operation for the connected device by the callback function. The UPL notify the operation (USB_YES/USB_NO) using the API function *R_usb_hstd_ClassCheckResult()* to the USB-BASIC-F/W. After issues the SET_CONFIGURATION request, USB-BASIC-F/W notifies to the operating UPL the device connection by callback function. When an operable driver is not registered, the USB-BASIC-F/W issues the SET_CONFIGURATION request to the connected device. In this case, the state transition is not notified to the UPL.

8.2.9 Host Battery Charging control (HBC)

HBC is the H/W control program for the target device that operates the CDP or the DCP defined by USB Battery Charging Specification Revision 1.2.

Each processing is executed as follows according to the timing of the USB-BASIC-F/W.

The VBUS is drived
The attach processing
The detach processing

Moreover, the processing is executed by the timing of PDDETINT interruption. There is no necessity for the control from UPL, and does not notify to UPL, either.

CDP and DCP serve as program exclusion. When DCP is operating, the USB communication cannot be used.

The processing flow of HBC is shown Figure 8.1.

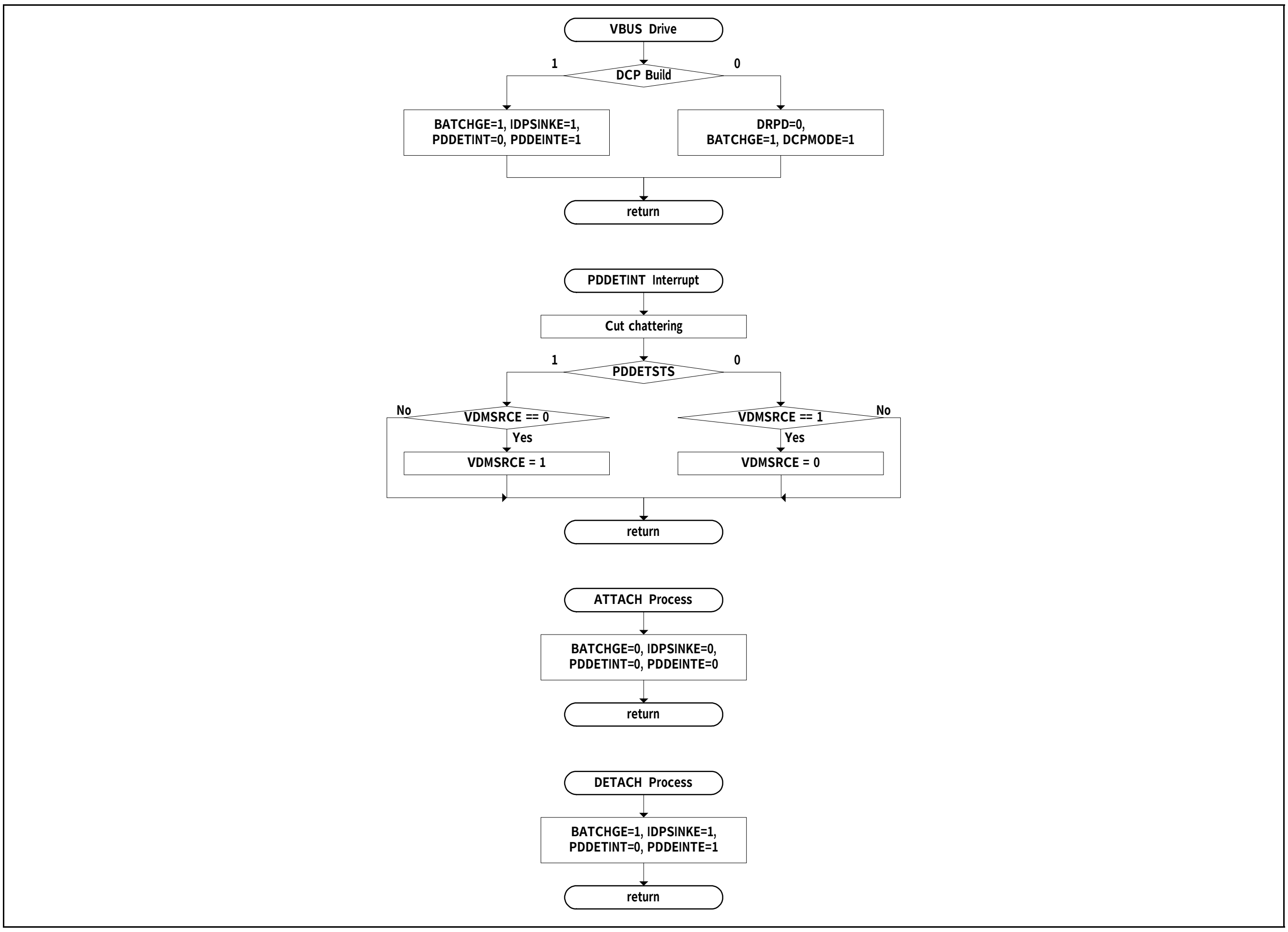

Figure 8.1 HBC processing flow

8.2.10 Notes on USB-BASIC-F/W

The USB-BASIC-F/W cannot enumerate to several devices simultaneously.

The USB-BASIC-F/W doesn't correspond to the multi configuration device.

The USB-BASIC-F/W doesn't correspond to the multi interface device.

When the UPL requests a suspension, interrupt the data transfer.

When the UPL receives resume completion or remote wakeup detection, resume the data transfer.

When detach is detected, the USB-BASIC-F/W stops the data transfer.

The USB-BASIC-F/W includes the following functions. Refer to the API function shown in Chapter 8.3 for more details.

(1) Control to disable the USB port.
(2) Change the USB state (suspend and resume).
(3) Clears the STALL pipe (cancel STALL to the connected device)
(4) Search Endpoint information from Descriptor.
(5) Interrupts the data transfer.
(6) Release the UPL

8.3 HCD API Function

The UPL request the H/W control using the API function. The API function is in the *r_usb_hdriverapi.c* file.

When using the HCD API function of USB-BASIC-F/W, include the header file according to the order shown in Table 8-5. Table 8-6 lists the HCD API functions.

Table 8-5 List of HCD API header file

File Name	Description	Notes
r_usb_ctypedef.h	Variable type definition	
r_usb_ckernelid.h	System header file	
r_usb_cdefusbip.h	Various definition for the USB driver	
r_usb_api.h	USB driver API function definition	

Table 8-6 List of HCD API Function

Function Name	Description	Notes
R_usb_hstd_HcdTask	HCD task	
R_usb_hstd_MgrTask	MGR task	
R_usb_hstd_HcdOpen	Start the MGR task and HCD task (Task initialization)	
R_usb_hstd_DriverRegistration	Register the UPL driver	
R_usb_hstd_DriverRelease	Release the UPL driver	
R_usb_hstd_TransferStart	Data transfer start request	
R_usb_hstd_TransferEnd	Data transfer forced end request	
R_usb_hstd_MgrChangeDevice State	Change the USB state of the connected device	
R_usb_hstd_ChangeDeviceState	Change the connected device state	
R_usb_hstd_DeviceInformation	Request the connected device state	
R_usb_hstd_ChkPipeInfo	Create pipe information from endpoint descriptor	
R_usb_hstd_ReturnEnuMGR	Enumeration continue request	
R_usb_hstd_SetPipeRegistration	Register setting of pipe information	

8.4 HCD Callback Function

The USB-BASIC-F/W notifies the USB state change and data transfer end to the UPL using the callback function. The UPL specifies the callback function when the API function is called or the driver is registered. When making the newly callback function, follow the order shown in Table 8-5 to include the header file in the same way when using the API function. Moreover, the HCD callback function list is shown in Table 8-7.

Table 8-7 List of HCD callback Function

Function Name	Description	Notes
*g_usb_HcdDriver[x].classcheck	Callback function of right or wrong operation to the connected device	
*g_usb_HcdDriver[x].statediagram	Callback function when USB state transition is detected	
* g_usb_LibPipe[pipe]->complete	Callback function when data transfer is occurred	
*g_usb_MgrCallback	Callback function of USB state transition end by API function	

8.5 Details for the API Function and Callback Function

Details for the API function and callback function are shown below.

R_usb_hstd_HcdTask

HCD task

Format

void R_usb_hstd_HcdTask(void)

Arguments

— —

Return Value

— —

Description

To call *usb_hstd_hcd_task()* function.

The *usb_hstd_hcd_task()* function is executed responded processing.

- Perform the USB control transfer.
- When the control transfer ends, call the callback function.
- When the USB state transition is detected, notify to the MGR task.

The *usb_hstd_hcd_task()* function performs the data transfer requested via the API function.

- When the data transfer ends, call the callback function specified by the API function.

The *usb_hstd_hcd_task()* function performs the USB state controlled(H/W control) from the MGR task.

- When the USB state changes, call the callback function.

Notes

1.Call this function in a loop where scheduler processing is performed.

2.Deadlock processing with while(1) when receiving the invalid message

Example

```
void main(void)
{
  usb_hsmpl_main_init();
  while( 1 )
  {
    if(R_usb_cstd_Scheduler() == USB_FLGSET )
    {
       R_usb_hstd_HcdTask();
       R_usb_hstd_MgrTask();
       usb_hsmpl_apl_task();
    }
  }
}
```

R_usb_hstd_MgrTask

MGR task

Format

void R_usb_hstd_MgrTask(void)

Arguments

— —

Return Value

— —

Description

To call *usb_hstd_mgr_task()* function

The *usb_hstd_mgr_task()* function manages the sequence of the USB state that the HCD task detected.

- Perform sequence control for enumeration.
- Perform sequence control for remote wakeup.
- Perform sequence control for detach and over current.
- When the end of the sequence control, call the USB state callback function registered by a user.

The *usb_hstd_mgr_task()* function manages the sequence of the USB state that via the API function.

- Perform sequence control for suspend or resume.
- Perform sequence control to enable or disable a port.
- Cancel STALL for the connected device.
  When the end of the sequence control, call the callback function specified by the API function.

Note

1. Call this function in a loop where scheduler processing is performed.

Example

```
void main(void)
{
  usb_hsmpl_main_init();
  while( 1 )
  {
   if(R_usb_cstd_Scheduler() == USB_FLGSET )
   {
      R_usb_hstd_HcdTask();
      R_usb_hstd_MgrTask();
      usb_hsmpl_apl_task();
   }
  }
}
```

R_usb_hstd_HcdOpen

HCD task start

Format

void R_usb_hstd_HcdOpen(void)

Arguments

— —

Return Value

— —

Description

Initialize global variables.

Note

1. To call the starting of USB-BASIC-F/W.

Example

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

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

  /* HCD driver open & registration */
  R_usb_hstd_HcdOpen();                     /* HCD task, MGR task open */
  usb_hsmpl_driver_registration();          /* Sample driver registration */

  /* USB-IP is set to the host */
  R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);
}
```

R_usb_hstd_DriverRegistration

Host device class driver (HDCD) registration

Format

void R_usb_hstd_DriverRegistration(usb_hcdreg_t * registinfo)

Argument

* registinfo	Class driver structure

Return Value

—	—

Description

Register the UPL to the USB-BASIC-F/W. Update the number of registered drivers controlled by the USB-BASIC-F/W and register the UPL information to the new area.

Notes

1. Call this function using the UPL for initialization.
2. Refer to Table 8-1 Member of usb_hcdreg_t Structure for information to be registered.

Example

```
void usb_hsmpl_driver_registration(void)
{
  usb_hcdreg_t  driver;

  /* Driver registration */
  driver.ifclass      = USB_IFCLS_VEN;         /* Vendor class */
  driver.classcheck   = &usb_hsmpl_class_check;
  driver.statediagram = &usb_hsmpl_open_close;
  R_usb_hstd_DriverRegistration(&driver);
}
```

R_usb_hstd_DriverRelease

Release host device class driver (HDCD)

Format

void R_usb_hstd_DriverRelease(uint8_t devclass)

Argument

devclass Device class (interface class code of USB2.0 specification)

Return Value

— —

Description

Release a device class driver registered to the USB-BASIC-F/W. Update the number of registered driver controlled by the USB-BASIC-F/W and leave the used area as an empty area.

Notes

1. When releasing a driver, call this function using the UPL.
2. Refer to Table 8-1 for the released information.
3. A typical interface class code is defined in *the r_usb_cdefusbip.h* file.

Example

```
ueb_er_t usb_smp_task(void)
{
  usb_hcdreg_t  driver;

    :
  R_usb_hstd_DriverRegistration(&driver);        /* Driver registration */
    :
  R_usb_hstd_DriverRelease(USB_IFCLS_HID);       /* Release HID class driver */

  /* Driver registration */
  driver.ifclass      = USB_IFCLS_VEN;           /* Vendor class */
  driver.classcheck   = &usb_hsmpl_class_check;
  driver.statediagram = &usb_hsmpl_open_close;
  R_usb_hstd_DriverRegistration(&driver);
    :

}
```

R_usb_hstd_TransferStart

Data transfer request

Format

usb_er_t R_usb_hstd_TransferStart(usb_utr_t * utr_table)

Argument

* utr_table	Structure of the data transfer

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap(The pipe is using.)

Description

Request the data transfer of each pipe. When the specified data size is satisfied, receiving a short packet, and an error occurs, the data transfer ends.
When the data transfer ends, call the callback function of the argument in the structure member. Remaining data length of transmission and reception, status, and information of transfer end are set in the argument of this callback function (*utr_table*).
When the data transfer is restarted with the same pipe, it is necessary to put the pipe status that does the communication demand this time as the pipe status (data toggle) that did the forwarding end last time together. Structure member (*utr_table.pipectr*) of the argument must set the pipe status. When the factors of USB reset or the clear STALL, etc. are generated, the pipe status should be initialized at "*DATA0*".
When a transfer start request is generated to the pipe during the data transfer, return USB_E_QOVR.

Notes

1. Refer to Table 8-4 Member of usb_utr_t Structure for the data transfer structure.
2. When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated.
3. The control transfer uses this API function.

Example

```
usb_utr_t   g_usb_HsmplTrnMsg[USB_TBL_MAX];
void usb_hvndr_data_transfer(usb_pipe_t pipe)
{
  /* PIPE Transfer set */
  g_usb_HsmplTrnMsg[pipe].pipenum = pipe;
  g_usb_HsmplTrnMsg[pipe].tranadr = g_usb_HsmplTrnPtr[pipe];
  g_usb_HsmplTrnMsg[pipe].tranlen = g_usb_HsmplTrnSize[pipe];
  g_usb_HsmplTrnMsg[pipe].pipectr = g_usb_HsmplPipeCtr[pipe];
  g_usb_HsmplTrnMsg[pipe].setup   = 0;
  g_usb_HsmplTrnMsg[pipe].complete   = (usb_cb_t)&usb_hvndr_transfer_result;
  R_usb_hstd_TransferStart((usb_utr_t *)&g_usb_HsmplTrnMsg[pipe]);
}
```

R_usb_hstd_TransferEnd

Data transfer forced end request

Format

usb_er_t R_usb_hstd_TransferEnd(usb_pipe_t pipe, usb_strct_t msginfo)

Arguments

pipe	Pipe number
msginfo	Communication status

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure
USB_E_QOVR	Overlap (transfer end request for the pipe during transfer end)

Description

Set the following values to argument msginfo and request forced end of the data transfer to the USB-BASIC-F/W.

- USB_DO_TRANSFER_STP: Data transfer forced end (The HCD calls back.)
- USB_DO_TRANSFER_TMO: Data transfer timeout (The HCD does not call back.)

The transfer end is notified using the callback function set when the data transfer is requested (*R_usb_hstd_TransferStart*) for the forced end due to *msginfo=USB_DO_TRANSFER_STP*. The remaining data length of transmission and reception, pipe control register value, and transfer *status = USB_DATA_STOP* are set using the argument of the callback function (*usb_utr_t*). When the forced end request to the pipe that doesn't execute data transfer is generated, USB_E_QOVR is returned.

Notes

1. When data transmission is interrupted, the FIFO buffer of the SIE is not cleared. When the FIFO buffer is transmitted in the double buffer, the data that has not been transmitted yet may be remained in the FIFO buffer.
2. When argument pipes are pipe 0 to pipe 3, the USB_E_QOVR error is returned. The USB_E_ERROR error is returned for pipe 8 or more.

Example

```
void  usb_smp_task(void)
{
  usb_er_t err;
    :

  /* Transfer end request */
  err = R_usb_hstd_TransferEnd(USB_PIPE4, USB_DO_TRANSFER_TMO);

  return err;
    :
}
```

R_usb_hstd_MgrChangeDeviceState

USB device state change request

Format

usb_er_t R_usb_hstd_MgrChangeDeviceStat(usb_cb_info_t complete, usb_strct_t msginfo, usb_strct_t keyword)

Arguments

complete	Callback function executed when the USB state changing ends.
msginfo	USB state to be changed
keyword	Descriptions are different depending on msginfo such as port number, device address, and pipe number.

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

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

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

Notes

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

Example

```
void  usb_smp_task(void)
{
  R_usb_hstd_MgrChangeDeviceState
   (usb_hsmpl_status_result, USB_DO_GLOBAL_SUSPEND, g_usb_hsmpl_Port);
}
```

R_usb_hstd_ChangeDeviceState

USB device state change request

Format

usb_er_t R_usb_hstd_ChangeDeviceState(usb_strct_t msginfo)

Argument

msginfo USB state to be changed

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

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

- USB_DO_INITHWFUNCTION
  Start the USB-IP and perform the software reset. Execute this function before the USB-BASIC-F/W starts.
- USB_DO_SETHWFUNCTION
  Set the the USB-IP as the USB host device. Execute this function after registering UPL.

Notes

1. This applicable function executes processing without the MGR task and the HCD task.

Example

```
void  usb_smp_task(void)
{
  R_usb_hstd_ChangeDeviceState(USB_DO_INITHWFUNCTION);
  R_usb_hstd_HcdOpen();                    /* HCD task open */
  usb_hsmpl_driver_registration();         /* Sample driver registration */
  R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);
    :
    :
}
```

R_usb_hstd_DeviceInformation

Obtaining the USB device state information

Format

void R_usb_hstd_DeviceInformation(usb_addr_t devaddr, uint16_t *table)

Argument

devaddr	Device address
*table	Table address to store the device information

Return Value

—	—

Description

Obtain the USB device information. Store the following information to an address specified to the argument (**table*).

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

Notes

1. Provide 4 word area for the argument **table*.
2. When specifying 0 to the device address, the following information is returned.

(1) When there is not a device during enumeration.
table[0] = USB_NOPORT, table[1] = USB_STS_DETACH
(2) When there is a device during enumeration.
table[0] = Port number, table[1] = USB_STS_DEFAULT

1.

Example

```
void  usb_smp_task(void)
{
  uint16_t  tbl[4];
    :
  /* Device information check */
  R_usb_hstd_DeviceInformation(devaddr, &tbl);
    :
}
```

R_usb_hstd_ChkPipeInfo

Setting the pipe information table

Format

usb_er_t	R_usb_hstd_ChkPipeInfo(uint16_t *table, uint8_t *descriptor)

Argument

table	Pipe information table
descriptor	Endpoint descriptor

Return Value

USB_DIR_H_IN	Set the IN endpoint.
USB_DIR_H_OUT	Set the OPUT endpoint.
USB_ERROR	Failure

Description

Analyze the endpoint descriptor and create the pipe information table for a pipe.
Fields where information is updated are as follows:

USB_TYPFIELD	USB_BULK .or. USB_INT
USB_SHTNAKFIELD	USB_SHTNAKON（USB_TYPFIELD == USB_DIR_H_IN 時）
USB_DIRFIELD	USB_DIR_H_IN .or. USB_DIR_H_OUT
USB_EPNUMFIELD	Endpoint number shown in the endpoint descriptor
USB_IITVFIELD	Interval counter (specified by 2 to the nth power)

Notes

1. Refer to Chapter 7.4 for the pipe information table.
2. Set the interval counter by 2 to the nth power.
3. Call this function by the driver check callback function processing if connected device can work.
4. When creating the information table using several pipes , to search the endpoint descriptor and call this function repeatedly to embed processing in the following cases:
   - When the interface includes several endpoints.
   - When communication for several endpoints in the multiple interfaces.

Example

```
void usb_hsmpl_pipe_info(uint8_t *table)
{
  usb_er_t      retval = USB_YES;
  uint16_t      *ptr;

  /* Check Endpoint Descriptor */
  ptr  = g_usb_hsmpl_DefEpTbl;
  for (; table[1] == USB_DT_ENDPOINT, retval != USB_ERROR; table += table[0],
ptr += USB_EPL)
  {
   retval = R_usb_hstd_ChkPipeInfo(ptr, table);
  }
  return retval;
}
```

R_usb_hstd_ReturnEnuMGR

Device class determination notification

Format

void R_usb_hstd_ReturnEnuMGR(uint16_t cls_result)

Argument

cls_result Right or wrong of operation of connecting device

Return Value

— —

Description

The driver check callback function notifies the operation (USB_YES/USB_NO) to the USB-BASIC-F/W using this API function. When USB_NO is returned using this function, the USB-BASIC-F/W check operation using other device class driver. Return USB_YES or USB_NO for cls_result.

Note

1. To call this function, when the driver check callback function is ended.

Example

```
void usb_hsmpl_enumaration(usb_tskinfo_t *mess)
{
   :
   retval = usb_hsmpl_pipe_info(g_usb_hsmpl_InterfaceTable,
              (uint8_t)g_usb_hsmpl_ConfigTable[2]);
   if( retval == USB_ERROR )
   {
     R_usb_hstd_ClassCheckResult(USB_NO);
   }
   else
   {
     R_usb_hstd_ClassCheckResult(USB_YES);
   }
}
```

R_usb_hstd_SetPipeRegistration

Set the pipe information to the register

Format

void	R_usb_hstd_SetPipeRegistation(uint16_t* table, uint16_t command)

Argument

table	Pipe information table
command	Command

Return Value

—	—

Description

- When the command is “USB_NO".
  All pipes specified with the pipe information table are unused set.
- When the command is “USB_YES".
  All pipes specified with the pipe information table are unused set.
  After it makes it to unused, the pipe information table follows and sets all pipes.

Notes

Example

```
void usb_hsmpl_open_close(uint16_t data1, uint16_t device_state)
{
  switch(device_state)
  {
    case  USB_DEVCONFIG:
        if( data1 == g_usb_hsmpl_Devaddr )
        {
            /* device address set */
            R_usb_hstd_SetPipeRegistration(g_usb_hsmpl_DefEpTbl, USB_YES);
            usb_hsmpl_task_operate(USB_SMPL_INIT);
        }
    break;
    case  USB_DEVDETACH:
             :
}
```

*g_usb_HcdDriver[x]. classcheck

Callback to check operation of connected device when enumerating

Format

void (*driver->classcheck)((uint16_t**)&table);

Arguments

table Device information to notify to the device driver

Return Value

— —

Description

The registered device class driver checks operation of the connected device. Refer to Table 8-2 Argument Array of classcheck for the argument information table.
Notify right or wrong of operation by the API function *R_usb_hstd_ReturnEnuMGR()*.

Notes

1. The USB-BASIC-F/W executes callback when received the Configuration Descriptor.
   (*driver->classcheck)((uint16_t**)&table);
2. When check ends, notify the result to the USB-BASIC-F/W using the API function of *R_usb_hstd_ReturnEnuMGR()*.

Example

Processing of the function that callback is done as an example.

```
void usb_hsmpl_class_check(uint16_t **table)
{
  g_usb_hsmpl_DeviceTable      = (uint8_t*)((uint16_t*)table[0]);
  g_usb_hsmpl_ConfigTable      = (uint8_t*)((uint16_t*)table[1]);
  g_usb_hsmpl_Devaddr          = (uint16_t)(*table[3]);
  g_usb_hsmpl_EnumerationSeq   = USB_SEQ_0;
  g_usb_hsmpl_Message.msginfo.w = USB_MSG_CLS_CHECKREQUEST;

  /* Class check of enumeration sequence move to class function */
  if( USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)&g_usb_hsmpl_Message) != USB_E_OK )
  {
   while(1);
  }
}
```

*g_usb_HcdDriver[x].statediagram

Callback when detecting the USB state transition

Format

void (*driver->statediagram)((uint16_t)data1, (uint16_t)device_state);

Arguments

data1	Device address
device_state	USB device state

Return Value

— —

Description

Generation for the USB state transition change is notified to the UPL.

1. Attach detection
   (*driver->statediagram)(USB_NO_ARG, USB_STS_ATTACH);
2. Issue USB reset signal
   (*driver->statediagram)(USB_NO_ARG, USB_STS_DEFAULT);
3. End of enumeration sequence processing
   (*driver->statediagram)(driver->devaddr, USB_STS_CONFIGURED);
4. Detach detection
   (*driver->statediagram)(g_usb_MgrDevAddr, USB_STS_DETACH);
5. Over current detection
   (*driver->statediagram)(driver->devaddr, USB_STS_OVERCURRENT);
6. End of remote wakeup sequence processing
   (*driver->statediagram)(g_usb_MgrDevAddr, USB_STS_WAKEUP);

Note

1. When the USB state is changed in API function *R_usb_hstd_ChangeDeviceState()* or *R_usb_hstd_MgrChangeDeviceState()* function, a callback concerned is not called.
2. The callback notification when it detects attach or issue USB reset, executed to all of registered the device class drivers.

Example

Processing of the function that callback is done as an example.

```
void usb_hsmpl_open_close(uint16_t data1, uint16_t device_state)
{
   case  USB_STS_DETACH:
      usb_hsmpl_transfer_end_all();
      R_usb_hvndr_DriverStop();
   break;
   case  USB_STS_ATTACH:
      R_usb_hvndr_DriverStart();
   break;
   case USB_STS_DEFAULT:
   case USB_STS_ADDRESS:
   break;
   case  USB_STS_CONFIGURED:
      g_usb_gmpl_DeviceAddr = data1;
      if( g_usb_gmpl_DeviceAddr != 0 )
      {
         R_usb_hstd_SetPipeRegistration(g_usb_hsmpl_DefEpTbl, USB_YES);
      }
      usb_hsmpl_tranfer_all();
   break;

   case  USB_STS_SUSPEND:
   break;
   case  USB_STS_RESUME:
   case  USB_STS_WAKEUP:
      usb_hsmpl_tranfer_all();
   break;
   case  USB_STS_OVERCURRENT:
   break;
  }
}
```

*g_usb_LibPipe[pipe]->complete

Callback for data transfer end

Format

void (*g_usb_LibPipe[pipe]->complete)(g_usb_LibPipe[pipe]);

Argument

g_usb_LibPipe Transfer message

Return Value

— —

Description

The data transfer end or forced is notified to the UPL.

Notes

1. A message when transfer is requested is returned. Table 8-8 lists the structure members update by the USB-BASIC-F/W.
2. Do not call back for the timeout (USB_DO_TRANSFER_TMO specified by the *R_usb_hstd_TransferEnd()* function).

Table 8-8 usb_utr_t Structure Members

Members	Update	Function	Notes
tranlen	Updated	The remaining data length is notified. (tranlen = tranlen of the forwarding - data length actually sent or received)	
status	Updated	The following transfer results are notified. USB_DATA_OK When the data transfer (transmission / reception) normally ends. USB_DATA_SHT When the data transfer ends with less than the specified data length. USB_DATA_OVR When the received data size is exceeded USB_DATA_STOP When the data transfer is forcibly ended USB_CTRL_END Control transfer end (PIPE0 only)	
pipectr	Updated	The pipe control register (PIPExCTR register) value is notified	
Other than above	Not updated	The contents requested to be transferred are stored.	

Example

Processing of the function that callback is done as an example.

```
void usb_hsmpl_transfer_result(usb_utr_t *mess)
{
  switch(mess->status)
  {
   case USB_DATA_OK:
   case USB_DATA_SHT:
   case USB_DATA_OVR:
       if ((mess->keyword == USB_PIPE4) || (mess->keyword == USB_PIPE5))
       {
           usb_hsmpl_DataTransfer(512, (uint8_t*)&g_usb_SmplTrnData);
       }
   break;
  }
}
```

*g_usb_MgrCallback

Callback when USB state update ends using the API function

Format

void (*g_usb_MgrCallback)((uint16_t)keyword, (uint16_t)msginfo);

Argument

keyword	The content is different according to msginfo like the port number, the device address, and the pipe number, etc.
msginfo	USB device state

Return Value

— —

Description

This function is the callback function to notify the API function *R_usb_hstd_MgrChangeDeviceState()* request end.

1. Port enable output end
   (*g_usb_MgrCallback)(g_usb_MgrPort, USB_STS_POWER);
2. Port disable output end
   (*g_usb_MgrCallback)(g_usb_MgrPort, USB_STS_PORTOFF);
3. Suspend sequence end
   (*g_usb_MgrCallback)(g_usb_MgrDevAddr, USB_STS_SUSPEND);
4. Resume sequence end
   (*g_usb_MgrCallback)(g_usb_MgrDevAddr, USB_STS_RESUME);
5. STALL cancelled for a pipe
   (*g_usb_MgrCallback)(g_usb_CurrentPipe, USB_STALL_SUCCESS);

Note

1. The suspension and the resume do the call backing in each device (Each device class screwdriver)

Example

9. System Definition

9.1 Scheduler

The USB-BASIC-F/W controls task using the scheduler function. Features of the scheduler are shown as follows.

1. The scheduler manages the requests generated by tasks or H/W in order of task ID.
2. When several requests are generated to the task, the scheduler processes the request in the FIFO structure.
3. USB-BASIC-F/W notifies the task the request end according to the callback function.
   Therefore, the UPL can use this system without modification of the scheduler.
4. Construct the task controlled by the scheduler as the function.
5. The scheduler does not dispatch and preempt to other tasks until exiting the task.
   Caution:
   Since the scheduler does not dispatch and preempt to tasks, response time of the USB Control transfer is not satisfied with the USB2.0 standard. Check if it is satisfied with the USB2.0 standard in a constructed system.

(1). The scheduler item defined as the user system is set.

Set the following items in the *r_usb_cKernelid.h* file.

```
#define    USB_IDMAX        ((uint8_t)5)    : Maximum value of task IDs*1 [9.1.1]
#define    USB_TABLEMAX     ((uint8_t)5)    : Number of messages storable in the task [9.1.2]
#define    USB_BLKMAX       ((uint8_t)5)    : Number of messages obtainable in a system [9.1.2]
```

*1: For the maximum number setting, add 1 to the highest ID number among the tasks to be used.

(2). Setup of task information

For each additional task, add the task ID and mailbox ID to the *r_usb_cKernelid.h* file. Keep the following points in mind when setting these items.

- Do not assign the same ID to more than one task.
- Set the same value assigned to the task ID and the mailbox ID.

The following settings are examples for vendor class drivers of the sample program.

```
#define  USB_PVEN_TSK   USB_TID_3        : Task ID
#define  USB_PVEN_MBX   USB_PVEN_TSK     : Mailbox ID
```

9.1.1 Task ID and Maximum Value of the Task ID

Set task IDs and the maximum value of the task ID. Do not overlap the values for the task ID. When setting the maximum value, set the maximum value + 1 of the task ID to be used. Set the UPL task ID to be used depending on the number to be used.

The task priority level becomes task ID order. The highest priority level becomes 0. In host mode, the task priorities set like "HCD task < MGR task < HCDC task". In peripheral mode, the task priorities set like "PCD task < PDCD task".

Use a macro defined in the *r_usb_cKernrlid.h* file for the task ID setting.

9.1.2 Number of Messages That Can be Stored with Task

The priority table stores processing requests from each task depending on priority. Set the maximum number where processing requests are stored.

9.1.3 Number of Messages That Can be Obtained in a System

Set the number of messages that can be obtained using R_USB_PGET_SND in a system. An area is saved until R_USB_REL_BLK is executed. When using all areas, an error is returned in R_USB_PGET_SND. Change the setting according to a system.

9.2 Scheduler Macro and Scheduler Function

Table 9-2 list the scheduler macro and the API function of the scheduler. The API function of the scheduler is in the *r_usb_cstd_libapi.c* file. When using the API function of the scheduler, include the header file according to the order listed in Table 9-1.

Table 9-1 List of Scheduler API header file

File Name	Description	Notes
r_usb_ctypedef.h	Variable type definition	
r_usb_ckernelid.h	System header file	
r_usb_cdefusbip.h	Various definition for the USB driver	
r_usb_api.h	USB driver API function definition	

Table 9-2 List of Scheduler Macro and Scheduler Function

Macro Name	File Name	Description	Notes
	R_usb_cstd_Scheduler	Scheduler processing	
R_USB_TRCV_MSG	R_usb_cstd_RecMsg	Check if execution is requested	
R_USB_SND_MSG	R_usb_cstd_SndMsg	Transmit processing requests to the priority table	
R_USB_ISND_MSG	R_usb_cstd_iSndMsg	Transmit processing requests to the priority table from interrupts	
R_USB_WAI_MSG	R_usb_cstd_WaiMsg	Execute R_USB_SND_MSG after executing a scheduler for specified times.	
R_USB_GET_SND	R_usb_cstd_PgetSend	After an area of a message is secured, R_USB_SND_MSG is execute	
R_USB_REL_BLK	R_usb_cstd_RelBlk	Release an area for a saved message	

R_usb_cstd_Scheduler

Scheduler processing

Format

uint8_t R_usb_cstd_Scheduler(void)

Argument

—	—

Return Value

USB_FLGSET	The task processing exists.
USB_FLGCLR	The task processing dose not exists.

Description

Perform scheduler processing.

Managing requests generated by the tasks and H/W according to the relative priority of the tasks

To call the task processing when the Return Value is USB_FLGSET.

Note

—

Example

```
void main(void)
{
  /* Initialized USBIP */
  usb_hsmpl_main_init();

  /* Sample main loop  */
  while( 1 )
  {
    if(R_usb_cstd_Scheduler() == USB_FLGSET ) /* Scheduler */
    {
        R_usb_hstd_HcdTask();    /* HCD Task */
        R_usb_hstd_MgrTask();    /* MGR Task */
        usb_hsmpl_apl_task();
    }
  }
}
```

R_usb_cstd_RecMsg

Check if execution is requested

Format

usb_er_t R_usb_cstd_RecMsg(uint8_t id, usb_msg_t** mess);

Argument

id	Task ID of received message
mess	Received message

Return Value

USB_E_OK	There is request processing
USB_E_ERROR	There is no request processing

Description

Confirmed the reception message.

When there is a reception message, the address of the message received to the argument "*mess*" is stored, and USB_E_OK is returned to the return value.

Note

—

Example

```
void usb_hsmpl_apl_task(void)
{
  usb_utr_t     *mess;
  usb_er_t      err;     /* Error code */

  /* Receive message */
  err = USB_TRCV_MSG( USB_HSMP_MBX, (usb_msg_t**)&mess );
  if( err != USB_E_OK )
  {
   return;
  }

  switch( mess->msginfo )
  {
  case USB_MSG_CLS_CHECKREQUEST:        /* Enumeration */
   usb_hsmpl_enumaration((usb_tskinfo_t *) mess);
   break;
  case USB_MSG_CLS_INIT:                /* Initialize */
   usb_hsmpl_initialized();
   break;
  case USB_MSG_CLS_TASK:
   usb_hsmpl_application(mess);
   break;
  default:
   break;
  }
}
```

R_usb_cstd_SndMsg

Transmit processing requests to the priority table

Format

usb_er_t	R_usb_cstd_SndMsg(uint8_t id, usb_msg_t* mess)

Argument

id	Task ID to send message.
mess	Transmitted message

Return Value

USB_E_OK	Message transmission completion
USB_E_ERROR	Task ID is not set
	Priority table is full (Can't send request to priority table)

Description

The message is stored in the priority level table.

Note

1. After the USB interruption of MCU is prohibited by the *usb_cpu_int_disable()* function, R_USB_ISND_MSG is called.

Example

```
void usb_hsmpl_check_request(uint16_t result)
{
  usb_er_t      err;

  g_usb_hsmpl_Message.msginfo  = USB_MSG_CLS_CHECKREQUEST;
  g_usb_hsmpl_Message.status   = result;

  /* Class check of enumeration sequence move to class function */
  err = USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)&g_usb_hsmpl_Message);
}
```

R_usb_cstd_iSndMsg

Transmit processing requests to the priority table from interrupts

Format

usb_er_t	R_usb_cstd_iSndMsg(uint8_t id, usb_msg_t* mess)

Argument

id	Task ID to send message
mess	Transmitted message

Return Value

USB_E_OK	Message transmission completion
USB_E_ERROR	Task ID is not set
	Priority table is full (Can't send request to priority table)

Description

When the message is transmitted in the interrupt handler blade, it uses it.

The message is stored in the priority level table.

Note

—

Example

```
void R_usb_hstd_InterruptHandler(void)
{
  usb_er_t          err;
  usb_intinfo_t     *ptr;

  /* Initialize Interrupt handler message */
  ptr = &g_usb_cstd_IntMsg[g_usb_cstd_IntMsgCnt];
  usb_hstd_check_interrupt_source(&ptr->keyword, &ptr->status);
  err = USB_ISND_MSG(USB_HCD_MBX, (usb_msg_t*)ptr);

  /* Renewal Message count  */
  g_usb_cstd_IntMsgCnt++;
  if( g_usb_cstd_IntMsgCnt == USB_INTMSGMAX )
  {
   g_usb_cstd_IntMsgCnt = 0;
  }

}
```

R_usb_cstd_WaiMsg

Execute R_USB_SND_MSG after executing a scheduler for specified times

Format

usb_er_t R_usb_cstd_WaiMsg(uint8_t id, usb_msg_t* mess, uint16_t times)

Argument

id	Task ID to send message
mess	Transmitted message address
times	executing a scheduler for specified times

Return Value

USB_E_OK	The message was able to be stored in the queue.
USB_E_ERROR	Task ID is not set
	The queue table is full (Can't send request to priority table)

Description

After a specified frequency executes the scheduler, R_USB_SND_MSG is executed.

Note

1. When the message notification is delayed, it uses it.
2. When the task of specifying is already in the waiting state, to register in the queue ignore the "*times*".
3. When R_USB_SND_MSG is executed and it responds USB_E_OK, the queue is updated in the FIFO structure.

   When two or more messages are registered in the queue, the message since the second is assumed to be " *times=1*" and the waiting frequency is counted again.
4. When R_USB_SND_MSG is executed and it responds USB_E_ERROR, the queue is not updated.

   The message that the count ends is assumed to be " *times=1*" and the waiting frequency is counted again.

Example

```
/* enumeration wait setting */
if( g_usb_HcdMgrMode[elseport] == USB_DEFAULT )
{
  err = USB_WAI_MSG(USB_MGR_MBX, (usb_msg_t*)g_usb_MgrMessage, 100);
  if( err != USB_E_OK )
  {
     USB_PRINTF1("### hMgrTask snd_msg error (%ld)\n", err);
  }
}
```

R_usb_cstd_PgetSend

After an area of a message is secured, R_USB_SND_MSG is execute

Format

usb_er_t R_usb_cstd_PgetSend(uint8_t id, usb_strct_t msginfo, usb_cbinfo_t complete, usb_strct_t keyword)

Argument

id	Task ID to send message.
msginfo	Message information
complete	Call-back function
keyword	Register the sub-information

Return Value

USB_E_OK	Message transmission completion
USB_E_ERROR	Task ID is not set
	Priority table is full (Can't send request to priority table)
	All the message areas are using it

Description

The area of the message is secured from the memory block.

The arguments (id, msginfo, complete, and keyword) are stored in the message of the secured area.

R_USB_SND_MSG is executed and the message is notified.

When R_USB_SND_MSG is executed and it responds USB_E_OK, the *flag* in the secured area is set up.

Note

1. The "*flag*" is an index of the secured area. Please specify it for an index number when the area is opened with R_USB_REL_BLK.

Example

```
void usb_hstd_detach(usb_port_t port)
{
  /* ATTCH interrupt enable */
  USB_CLR_PAT(DVSTCTR0, (uint16_t)(USB_RWUPE | USB_USBRST | USB_RESUME |
USB_UACT));
  usb_hstd_attch_enable(port);
  USB_PGET_BLK
   (USB_MGR_MBX, USB_DO_DETACH, &usb_cstd_dummy_function ,(uint8_t)port);
}
```

R_usb_cstd_RelBlk

Release an area for a saved message

Format

usb_er_t R_usb_cstd_RelBlk(uint8_t blk_num)

Argument

blk_num	An index number when the area is opened

Return Value

USB_E_OK	The area is released
USB_E_ERROR	The area is not released

Description

The argument "*blk_num*" is assumed to be an index, and the "*flag*" in the area to be released is retrieved.

When the "*blk_num*" is corresponding to the "*flag*", the area is released.

Note

—

Example

```
void R_usb_pstd_PcdTask(usb_vp_int_t stacd)
{
  usb_tskinfo_t    *mess;
  /* Error code */
  usb_er_t      err;

  err = USB_TRCV_MSG(USB_PCD_MBX, (usb_msg_t**)&mess, (usb_tm_t)10000);
  if( (err != USB_E_OK) )
  {
    return;
  }

  g_usb_PcdMessage = (usb_tskinfo_t*)mess;

  switch( g_usb_PcdMessage->msginfo )
  {
    case USB_DO_REMOTEWAKEUP:
    case USB_PCD_DP_ENABLE:
    case USB_PCD_DP_DISABLE:
       (*g_usb_PcdCallback)((uint16_t)USB_NO_ARG, g_usb_PcdMessage->msginfo);
       USB_REL_BLK(g_usb_PcdMessage->flag);
    break;
    default:
    break;
  }
}
```

9.3 Common Library Function

Table 9-3 lists the common library API function that can be used by the user for host mode or peripheral mode commonly. Common library function is in the common library API function *r_usb_cstdapi.c* file. When using the common library API function, include *r_usb_api.h*.

Table 9-3 List of Common Library Function

Function Name	Description	Notes
R_usb_cstd_SetBufPipe0	Set PID of pipe 0 to BUF.	

R_usb_cstd_SetBufPipe0

Set PID of pipe 0 to BUF.

Format

void R_usb_cstd_SetBufPipe0(void)

Argument

— —

Return Value

— —

Description

Set PID of pipe 0 to BUF.

Note

—

Example

```
void usb_pstd_set_ccpl(void)
{
  R_usb_cstd_SetBufPipe0();              /* Request ok */
  USB_SET_PAT(DCPCTR, USB_CCPL);         /* Status stage start */
}
```

10. Restrictions

USB-BASIC-F/W includes the following restrictions.

1. Methods to use pipes is restricted using the pipe information setting function.
   - Use the transaction counter using the SHTNAK function for received pipes.
2. Members with different types comprise a structure.
   (An address misalignment of structure members may occur depending on compilers.)
3. Prepare the UPL by the user.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
Rev.1.00	Apr. 25, 2011	—	First edition issued
Rev.2.00	Nob. 30, 2012	—	Revision of the document by firmware update
Rev.2.01	Mar. 26, 2013	—	Added about IAR edition.

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord 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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

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.

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2