SH7786 Group

USB 2.0 Host Controller IP with Renesas EHCI Support
USB Basic Firmware µITRON Version

R01AN1452EJ0103
Rev.1.03
Dec 7, 2012

Introduction

This document is the user's manual for "USB 2.0 Host Controller IP with Renesas EHCI Support, USB Basic Firmware µITRON Version," a sample program for USB interface control that uses the USB 2.0 host controller IP with Renesas EHCI support.

Target Device

SH7786

Contents

1.   Document Overview

1.1   Overview

This document is the user's manual for "USB 2.0 Host Controller IP with Renesas EHCI Support, USB Basic Firmware µITRON Version," a sample program for USB interface control that uses the USB 2.0 host controller IP with Renesas EHCI support.

The USB basic firmware for the USB 2.0 host controller IP with Renesas EHCI support is compatible with µITRON.

This document is intended to be used in conjunction with the associated data sheet.

1.2   Related Documents

1. Universal Serial Bus Revision 2.0 Specification
    [http://www.usb.org/developers/docs/]
2. SH778 Group User's Manual(Hardware part)
    This document can be downloaded from the Renesas Electronics Web site.
*   Renesas Electronics Web site
    [http://www.renesas.com/]
*   USB Devices page
    [http://www.renesas.com/prod/usb/]

1.3   List of Terms

The following terms and abbreviations are used in this document.

USB	: Universal serial bus
EHCI	: Enhanced host controller interface
OHCI	: Open host controller interface
USB-BASIC-F/W	: USB basic firmware for USB 2.0 host controller IP with Renesas EHCI support (µITRON)
µITRON	: USB basic firmware for µITRON system
HEW	: High-performance embedded workshop
HCD	: Host control driver of USB-BASIC-F/W
MGR	: Peripheral device state manager of HCD
HDCD	: Host device class driver (device driver and USB class driver)
HUBCD	: Hub class sample driver
APL	: Application program

2.   Overview

2.1   Features of USB-BASIC-F/W

USB-BASIC-F/W offers the following features.

- Support for host function operation
- Inclusion of sample program for control transfer (enumeration)
- Inclusion of sample program for device attach/detach processing
- Inclusion of sample program for suspend/resume processing
- Inclusion of HUBCD sample program
- Ability to load multiple device class drivers without having to customize USB-BASIC-F/W (Multiple device class drivers can be registered, up to the maximum number of devices that can be connected.)

The user should prepare the following functions to match the system under development.

- Handler for overcurrent detection at USB cable connection
- Descriptor parser
- Device class drivers

2.2   Development Goal

USB-BASIC-F/W was developed to accomplish the following goal.

- To simplify the development by the user of USB communication programs employing the USB 2.0 host controller IP with Renesas EHCI support.

2.3   Functions

USB-BASIC-F/W provides the following functions.

- Enumeration of low-speed, full-speed, and high-speed devices
- USB connector attach/detach, suspend/resume, and USB bus reset processing
- Control transfer via pipe 0
- Data transfer (bulk transfer, interrupt transfer, and isochronous transfer) via pipes 1 to 30
- Transfer error determination

2.4   Task Structure

USB Basic Firmware support comprises a host driver that implements the host function, a host manager that manages device states, an HCD transfer task that manages communication with USB devices, an HCD System task that manages the USB ports, a hub class driver that controls a device connected to a down port of the USB hub, and an application.

The host driver starts hardware control according to messages from the various tasks. It also notifies the various tasks of hardware control end, the processing result, and hardware requests.

The host manager is a sample program that performs state management and enumeration for the device connected to the root port. In addition, the host manager sends a message to the host driver or hub class driver by means of the device address when the application updates the device state. The hub class driver is a sample program that performs enumeration and state management for devices connected to the down ports of the USB hub.
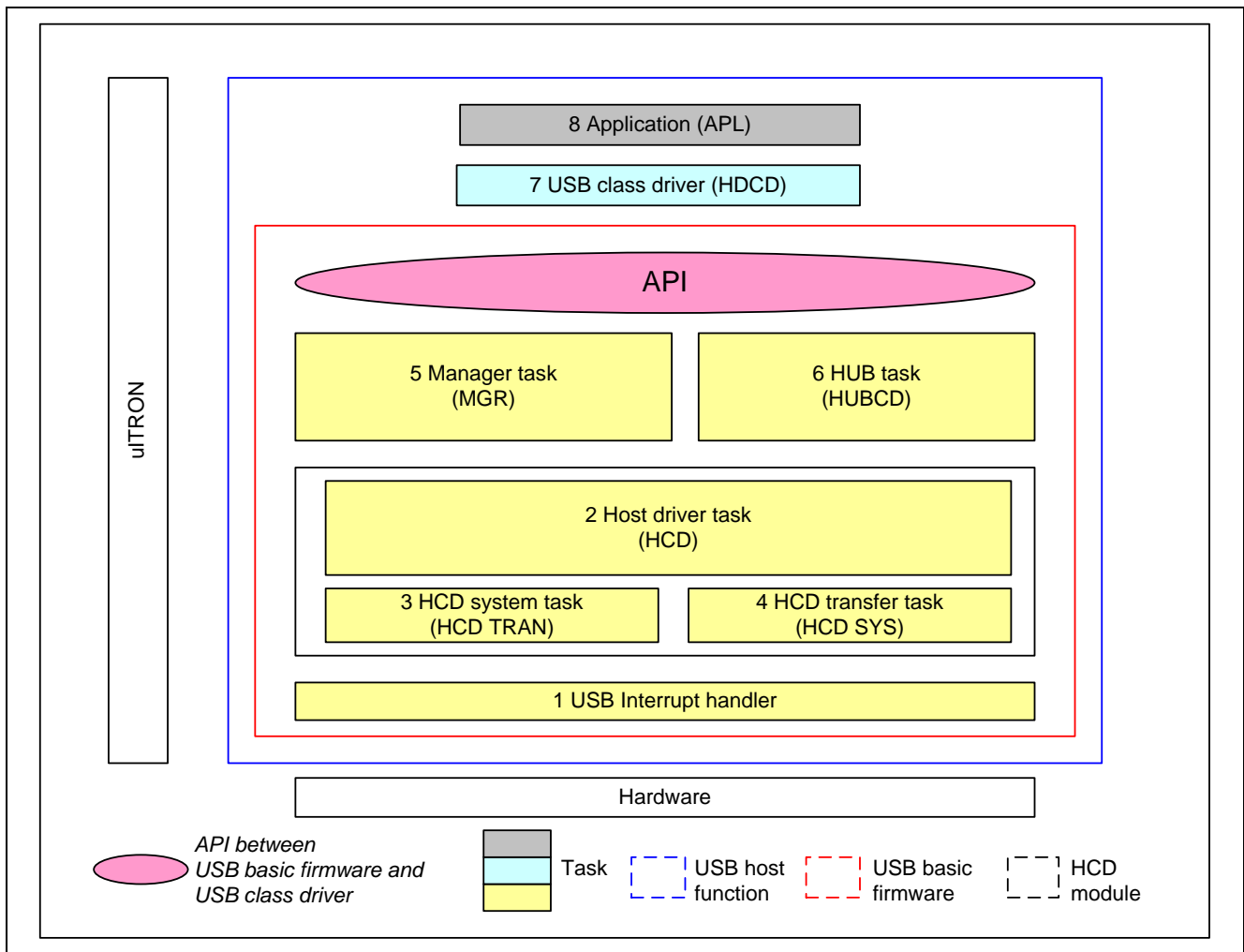
Figure 2.1   Task Structure of USB-BASIC-F/W

Table 2.1   Overview of Task Functions

No.	Module	Function
1	USB interrupt handler	USB interrupt handler (USB packet transmit/receive end and special signal detection)
2	Host control driver (HCD)	Host communication control
3	HCD Transfer (HCD_TRN)	• Host function hardware control • Host transaction management
4	HCD System (HCD_SYS)	• Host function hardware control • USB port state management • Bus reset control
5	Manager (MGR)	• Device state management • Enumeration • HCD/HUBCD control message determination
6	Hub class driver (HUBCD)	• HUB down port device state management • HUB down port enumeration
7	Host device class driver (HDCD)	Execution of processing dependent on the device class (prepared by the user to match the system)
8	Application (APL)	Execution of application (prepared by the user to match the system)

2.5   Outline Flowchart

USB-BASIC-F/W consists of tasks comprising control functions for transmitting and receiving USB data.

When an interrupt occurs, a message is sent to the HCD Transfer task. When the HCD Transfer task receives a message from the USB interrupt handler, it determines the interrupt source and executes the appropriate processing. (Outline flowcharts of the individual tasks appear in the next and subsequent sections.)
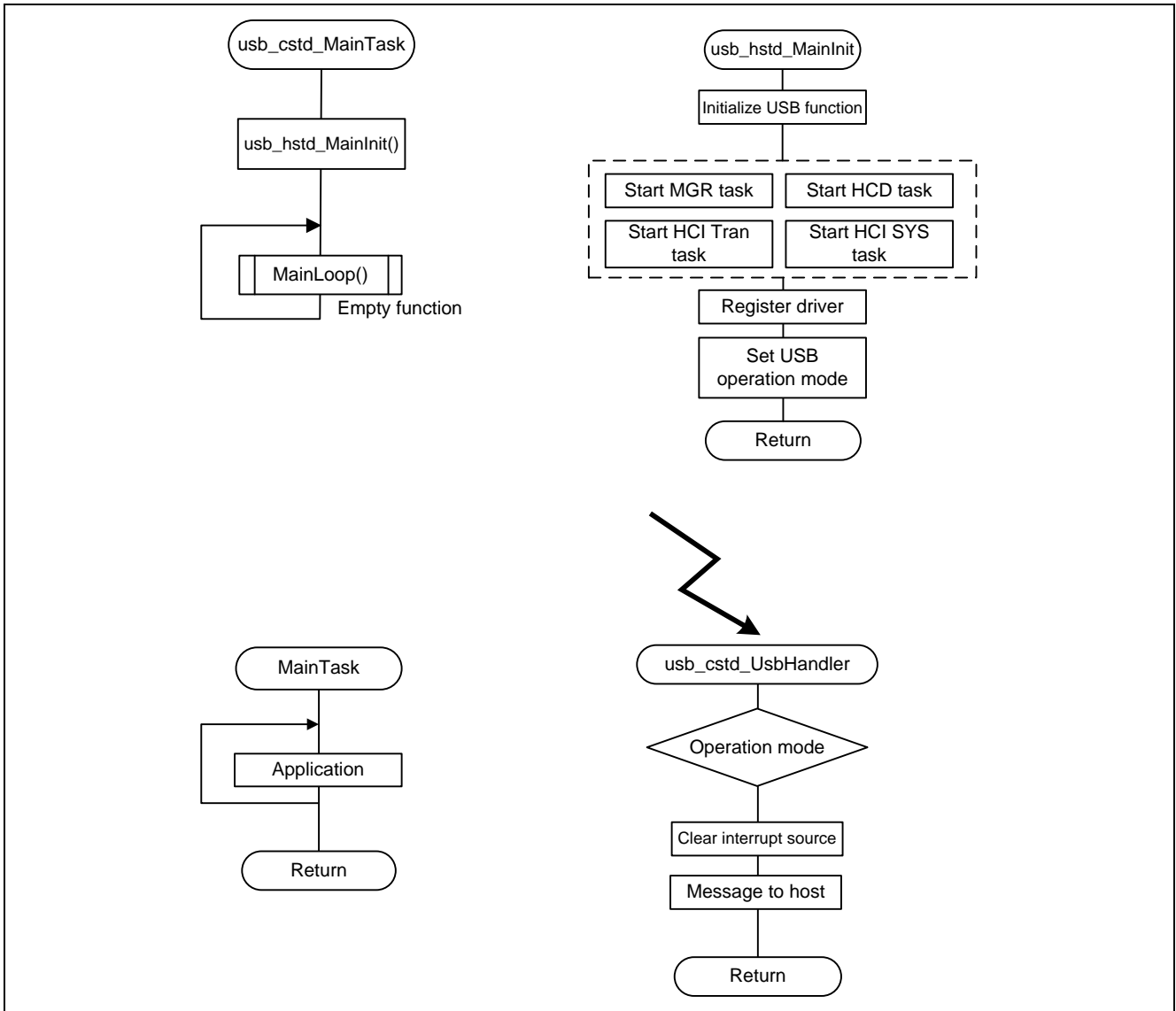

Figure 2.2   Outline Flowchart

2.6   µITRON Task Association Chart

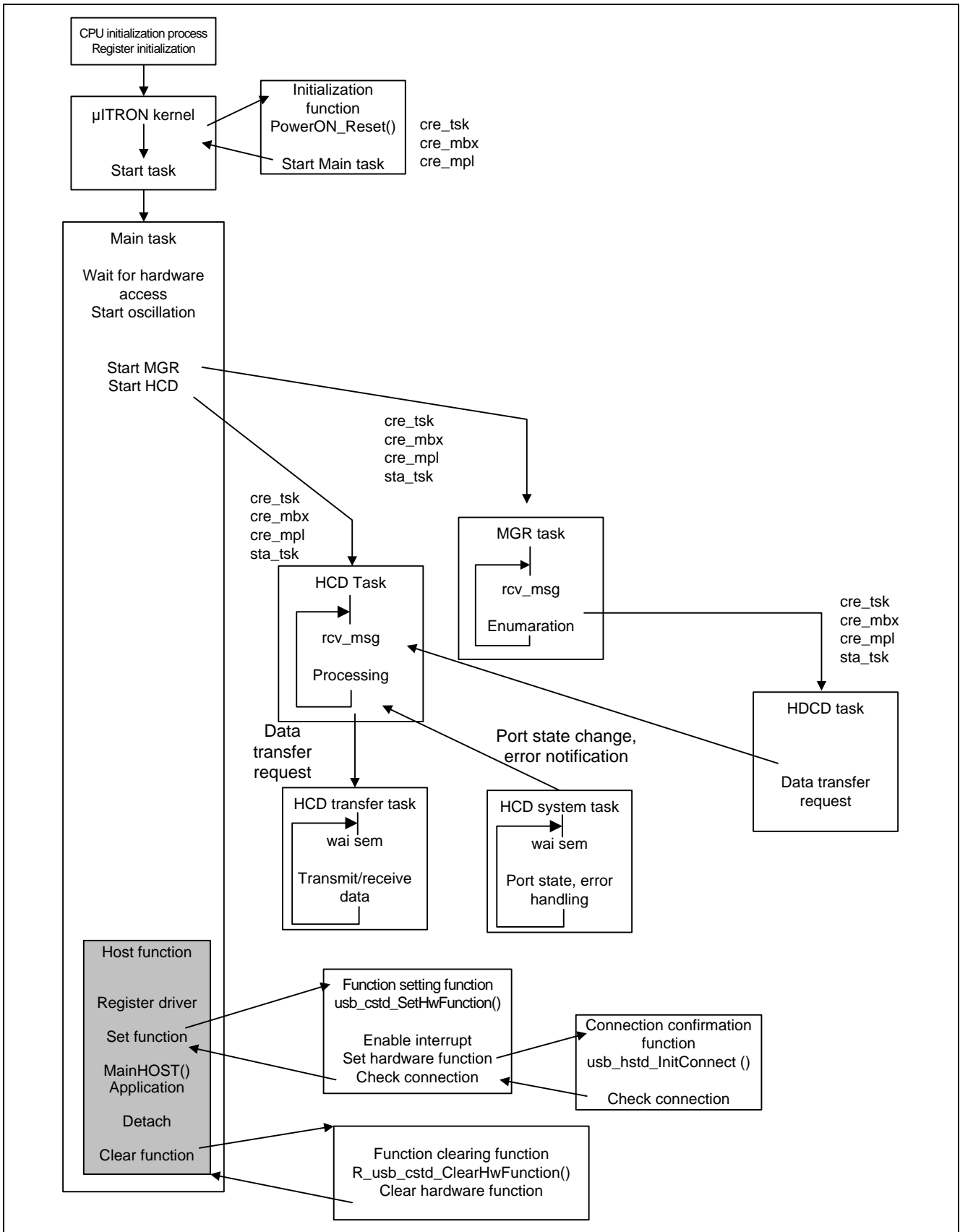A task association chart for USB-BASIC-F/W is shown below.

Figure 2.3   Task Association Chart

3. Using USB-BASIC-F/W

3.1 Overview

USB-BASIC-F/W can be configured as a USB driver adapted to the user's system by making changes to the scheduler macros (**r_usb_cItron.h, r_usb_cMacItron.h**), the user information (**r_usb_cDefUsrPb.h, r_usb_cDefUsr.h**), and the OS settings (**r_usb_cKernelId.h, main.c, r_usb_HSMPL_apl.c**), then adding HDCD to the usb_hstd_MainLoop function.

3.2 Making Changes to USB-BASIC-F/W

Changes must be made to the program and header files listed below before USB-BASIC-F/W can be used.

1. Source code is provided for main.c and the sample functions of the USB 2.0 host controller IP with Renesas EHCI support, but the following changes must be made to match the user's system.
   — Initialization function for the MCU used for control, interrupt handler, interrupt control functions, etc. (See table 3.1.)
   — Adjustment of the wait time specified for the designated duration wait functions (usb_cstd_DelayXms() function, usb_cstd_Delay1us() function)
   Loop processing or the like is used to wait for the specified amount of time. Adjust the wait duration to match the system under development by, for example, changing the loop count.
   USB-BASIC-F/W uses the USB interrupt disable function (usb_cstd_IntDisable() function) to disable USB interrupts and the USB interrupt enable function (usb_cstd_IntEnable() function) to enable them. The settings of these functions should be altered as needed to match the specifications of the MCU.
2. Some files must be customized by the user.
   Refer to 5., User-Defined Information, and change the user settings as needed.
3. Debug Information Output Function
   Output of debug information can be enabled or disabled by means of a setting in the **r_usb_cMacPrint.h** file. (It is possible to output debug information by preparing a serial driver, etc.)

Table 3.1   List of Functions

Type	Function	Description
void	usb_cstd_TargetInit(void)	Initializes the system
void	usb_cstd_UsbIntHand(void)	USB interrupt handler
void	usb_cstd_UsbintInit(void)	Enables USB interrupts
void	usb_cstd_IntEnable(void)	Enables USB interrupts
void	usb_cstd_IntDisable(void)	Disables USB interrupts
void	usb_cstd_Delay1us(uint16_t time)	Waits 1 us
void	usb_cstd_DelayXms(uint16_t time)	Waits 1 ms
void	usb_cstd_VbusControl(uint16_t port, uint16_t command)	VBUS on/off control

3.3 Preparing HDCD

HDCD must be prepared to match the user's system in order to run USB-BASIC-F/W. Perform the following steps to prepare HDCD.

- Registration of HDCD (Use R_usb_hstd_DriverRegistration() to register HDCD in MGR.)
- Preparation of class checking routine (It is necessary to perform class checking during enumeration.)
- Pipe settings (Use R_usb_hstd_SetPipeRegistration() to specify the pipes used by HDCD.)

In addition to the above, prepare any routines required by the system under development.

3.4   Note

The user should customize the provided source code as necessary to accommodate cases in which class stipulation or issuing of vendor-specific requests is necessary, considerations related to communication speed or program size, or individual user interface settings.

Note: USB-BASIC-F/W contains no functionality to verify the integrity of USB data communication. When applied in the user's system, it is up to the user to verify its operation and to confirm its ability to connect with a variety of devices.

4. User-Defined Macros

4.1 Overview

USB-BASIC-F/W includes µITRON system call and debug output macros. The user can create a customized executable file by modifying the header files of these macros. Make changes to the macros as necessary to match the system under development. The two macros listed below are provided. User settings for the µITRON system call macro are defined in **r_usb_cMacItron.h** and for the debug output macro in **r_usb_cMacPrint.h**.

1. µITRON system call macro
2. Debug output macro

4.2 µITRON System Call Macro

This is the µITRON system call macro.

Customize this macro as necessary to match the version of ITRON to be used.

Note that the ITRON macro is redefined for USB-BASIC-F/W by the **r_usb_cMacItron.h** file, and the ITRON macro is defined by conditional compilation even if ITRON is not used. This file should therefore also be customized to match the version of ITRON to be used.

```
#define  USB_CRE_TSK(ID,INFO)          cre_tsk( (USB_ID_t)ID, (USB_TSK_t*)INFO )
#define  USB_DEL_TSK(ID)               del_tsk( (USB_ID_t)ID )
#define  USB_STA_TSK(ID,CODE)          sta_tsk( (USB_ID_t)ID, (USB_VI_t)CODE )
#define  USB_ACT_TSK(ID)               act_tsk( (USB_ID_t)ID )
#define  USB_TER_TSK(ID)               ter_tsk( (USB_ID_t)ID )
#define  USB_EXT_TSK()                 ext_tsk( )
#define  USB_REF_TST(ID, STS)          ref_tst( (USB_ID_t)ID, (USB_RTST_t*)STS )


#define  USB_DLY_TSK(TIME)             dly_tsk( (USB_RT_t)TIME )


#define  USB_CRE_MBX(ID, INFO)         cre_mbx( (USB_ID_t)ID, (USB_MBX_t*)INFO )
#define  USB_DEL_MBX(ID)               del_mbx( (USB_ID_t)ID )
#define  USB_SND_MSG(ID, MESS)         snd_mbx( (USB_ID_t)ID, (USB_MSG_t*)MESS )
#define  USB_ISND_MSG(ID, MESS)        isnd_mbx( (USB_ID_t)ID, (USB_MSG_t*)MESS )
#define  USB_RCV_MSG(ID, MESS)         rcv_mbx( (USB_ID_t)ID, (USB_MSG_t**)MESS )
#define  USB_PRCV_MSG(ID, MESS)        prcv_mbx( (USB_ID_t)ID, (USB_MSG_t**)MESS )
#define  USB_TRCV_MSG(ID, MESS, TM)    trcv_mbx( (USB_ID_t)ID, (USB_MSG_t**)MESS,
                                                (USB_TM_t)TM )


#define  USB_CRE_MPL(ID, INFO)         cre_mpf( (USB_ID_t)ID, (USB_MPL_t*)INFO )
#define  USB_DEL_MPL(ID)               del_mpf( (USB_ID_t)ID )
#define  USB_PGET_BLK(ID, BLK)         pget_mpf( (USB_ID_t)ID, (USB_MH_t*)BLK )
#define  USB_IPGET_BLK(ID, BLK)        ipget_mpf( (USB_ID_t)ID, (USB_MH_t*)BLK )
#define  USB_REL_BLK(ID, BLK)          rel_mpf( (USB_ID_t)ID, (USB_MH_t)BLK )
```

#define USB_CRE_SEM(ID, INFO)	cre_sem((USB_ID_t)ID, (USB_SEM_t*)INFO)
#define USB_WAI_SEM(ID)	wai_sem((USB_ID_t)ID)
#define USB_POL_SEM(ID)	pol_sem((USB_ID_t)ID)
#define USB_SIG_SEM(ID)	sig_sem((USB_ID_t)ID)
#define USB_DEL_SEM(ID)	del_sem((USB_ID_t)ID)
#define USB_ISIG_SEM(ID)	isig_sem((USB_ID_t)ID)
#define USB_CRE_ALM(ID, INFO)	cre_alm((USB_ID_t)ID, (USB_ALM_t*)INFO)
#define USB_STA_ALM(ID, TIME)	sta_alm((USB_ID_t)ID, (USB_RT_t)TIME)
#define USB_STP_ALM(ID)	stp_alm((USB_ID_t)ID)
#define USB_DEL_ALM(ID)	del_alm((USB_ID_t)ID)

4.3   Debug Information Output Macro

This macro outputs debug information to the UART or to a display device of some sort. A serial driver or display device driver is required in order to use it.

#define USB_SPRINTF0(FORM)	fprintf(stderr,FORM)
#define USB_SPRINTF1(FORM,x1)	fprintf(stderr,FORM,x1)
#define USB_SPRINTF2(FORM,x1,x2)	fprintf(stderr,FORM,x1,x2)
#define USB_SPRINTF3(FORM,x1,x2,x3)	fprintf(stderr,FORM,x1,x2,x3)
#define USB_SPRINTF4(FORM,x1,x2,x3,x4)	fprintf(stderr,FORM,x1,x2,x3,x4)
#define USB_SPRINTF5(FORM,x1,x2,x3,x4,x5)	fprintf(stderr,FORM,x1,x2,x3,x4,x5)
#define USB_SPRINTF6(FORM,x1,x2,x3,x4,x5,x6)	fprintf(stderr,FORM,x1,x2,x3,x4,x5,x6)
#define USB_SPRINTF7(FORM,x1,x2,x3,x4,x5,x6,x7)	fprintf(stderr,FORM,x1,x2,x3,x4,x5,x6,x7)
#define USB_SPRINTF8(FORM,x1,x2,x3,x4,x5,x6,x7,x8)	fprintf(stderr,FORM,x1,x2,x3,x4,x5,x6,x7,x8)
#define USB_PRINTF0(FORM)	printf(FORM)
#define USB_PRINTF1(FORM,x1)	printf(FORM,x1)
#define USB_PRINTF2(FORM,x1,x2)	printf(FORM,x1,x2)
#define USB_PRINTF3(FORM,x1,x2,x3)	printf(FORM,x1,x2,x3)
#define USB_PRINTF4(FORM,x1,x2,x3,x4)	printf(FORM,x1,x2,x3,x4)
#define USB_PRINTF5(FORM,x1,x2,x3,x4,x5)	printf(FORM,x1,x2,x3,x4,x5)
#define USB_PRINTF6(FORM,x1,x2,x3,x4,x5,x6)	printf(FORM,x1,x2,x3,x4,x5,x6)
#define USB_PRINTF7(FORM,x1,x2,x3,x4,x5,x6,x7)	printf(FORM,x1,x2,x3,x4,x5,x6,x7)
#define USB_PRINTF8(FORM,x1,x2,x3,x4,x5,x6,x7,x8)	printf(FORM,x1,x2,x3,x4,x5,x6,x7,x8)

Comment out the following lines, as shown, to disable output of debug information.

//#define	USB_DEBUGSIO_PP	/* enable serial out (printf) */
//#define	USB_DEBUGLCD_PP	/* enable display out (fprintf) */

5.   User-Defined Information

5.1   Overview

USB-BASIC-F/W enables the user to create a customized executable file by modifying the user-defined system information file (**r_usbc_cDefUsrPb.h**), the user-defined information file (**r_usb_cDefUsr.h**), the EHCI user-defined information file (**r_usb_hEhciDefUsr.h**), and the OHCI user-defined information file (**r_usb_hOhciDefUsr.h**). Make changes to the following items as necessary to match the system under development.

Note: The values of items other than those listed below are fixed and should not be changed.

5.2   User-Defined System Information File (r_usb_cDefUsrPb.h)

1. USB port specification
2. Specification of OS create system call support
3. CPU byte endian specification

5.2.1   USB Port Specification

Specify the number of USB ports as one of the following two options.

1) USB_1PORT_PP: Use one USB port.
2) USB_2PORT_PP: Use two USB ports.
Example: Using one USB port
         #define   USB_PORTSEL_PP   USB_1PORT_PP

5.2.2   Specification of OS Create System Call Support

Specify whether or not to support the OS's create system call.

1) USB_OS_CRE_USE_PP: Support the create system call.
2) USB_OS_CRE_NOTUSE_PP: No not support the create system call.
Example: Create system call not supported
         #define   USB_OS_CRE_MODE_PP   USB_OS_CRE_NOTUSE_PP

5.2.3   CPU Byte Endian Specification

Specify the CPU's endian mode as one of the following two options. This item specifies the endian order used when transmitting and receiving data. (It is defined in **r_usb_cTypedef.h.**)

1) USB_BYTE_LITTLE_PP: Little endian (least significant byte first)
2) USB_BYTE_BIG_PP: Big endian (most significant byte first)
Example: Using the little endian CPU byte endian specification
         #define   USB_CPUBYTE_PP   USB_BYTE_LITTLE_PP

5.3   User-Defined Information File (r_usb_cDefUsr.h)

1. Control read data buffer size
2. Initial value of device address
3. Number of hub down ports

5.3.1   Control Read Data Buffer Size

Specify the data buffer size used when receiving data by control read transfer.

Example: 20-byte device descriptor and 256-byte configuration descriptor
         #define   USB_DEVICESIZE 20u
         #define   USB_CONFIGSIZE 256u

5.3.2 Device Address

Specify the device address of the device connected to PORT0.

Example: Starting from device address 1
        #define USB_DEVICEADDR 1u

5.3.3 Number of Hub Down Ports (when Host Function Selected)

Specify the number of down ports that can be connected to the hub.

Example: Allowing connection to the hub of up to four down ports
        #define USB_HUBDOWNPORT   4u

5.4 EHCI User-Defined Information File (r_usb_hEhciDefUsr.h)

1. Specification of EHCI hardware address
2. Specification of EHCI periodic frame list size
3. Specification of EHCI queue head data structure maximum memory size
4. Specification of EHCI qTD data structure maximum memory size
5. Specification of EHCI iTD data structure maximum memory size
6. Specification of EHCI siTD data structure maximum memory size
7. Specification of EHCI iTD data structure maximum data transfer size
8. Specification of EHCI timeout duration

5.4.1 Specification of EHCI Hardware Address

Specify the reference address for accessing the EHCI hardware. The addresses of the various registers are specified by offsets from the reference address.

Example: Specifying the address 0x000A0000
        #define USB_EHCI_BASE   0x000A0000

5.4.2 Specification of EHCI Periodic Frame List Size

Specify a value of 256, 512, or 1024 as the EHCI periodic frame list size. The scheduling amplitude for periodic transfers changes according to the size specified.

Example: Specifying a size of 256
        #define USB_EHCI_PFL_SIZE   256

5.4.3 Specification of EHCI Queue Head Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI queue head data structure. The queue head data structure must be able to accommodate the number of endpoint pipes used for control transfers, bulk transfers, and interrupt transfers. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 16
        #define USB_EHCI_NUM_QH   16

5.4.4   Specification of EHCI qTD Data Structure Maximum Memory Size

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

The qTD data structure is used as a transfer management descriptor linked to the queue head in control transfers, bulk transfers, and interrupt transfers. The maximum data size that can be transferred with a single qTD is 20,480 bytes. In addition, control transfers require one qTD for each Setup stage, Data stage, and Status stage. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 256
          #define USB_EHCI_NUM_QTD   256

5.4.5   Specification of EHCI iTD Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI iTD (high-speed isochronous transfer descriptor) data structure. The iTD data structure must be able to accommodate the number of endpoint pipes used for high-speed isochronous transfers. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 4
          #define USB_EHCI_NUM_ITD   4

5.4.6   Specification of EHCI siTD Data Structure Maximum Memory Size

Specify the maximum memory size for the EHCI siTD (split transaction isochronous transfer descriptor) data structure. The siTD data structure must be able to accommodate the number of endpoint pipes used for split transaction isochronous transfers. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 4
          #define USB_EHCI_NUM_SITD   4

5.4.7   Specification of EHCI iTD Data Structure Maximum Data Transfer Size

Specify the maximum data transfer size for the EHCI iTD data structure. The transfer size value specifies the maximum transfer size for a single high-speed isochronous transfer (one transaction). The largest supported maximum data size setting is 1024. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 512
          #define USB_EHCI_ITD_DATA_SIZE   512

5.4.8   Specification of EHCI Timeout Duration

Specify the EHCI timeout duration in msec units.

Example: Setting a value of 3
          #define USB_EHCI_TIMEOUT   3

5.5 OHCI User-Defined Information File (r_usb_hOhciDefUsr.h)

1. Specification of OHCI hardware address
2. Specification of OHCI endpoint data structure maximum memory size
3. Specification of OHCI endpoint descriptor data structure maximum memory size
4. Specification of OHCI transfer descriptor data structure maximum memory size
5. Specification of OHCI maximum isochronous device count
6. Specification of OHCI maximum isochronous data transfer size
7. Specification of OHCI maximum isochronous frame count
8. Specification of OHCI timeout duration

5.5.1 Specification of OHCI Hardware Address

Specify the reference address for accessing the OHCI hardware. The addresses of the various registers are specified by offsets from the reference address.

Example: Specifying the address 0x000A0000
   #define USB_OHCI_BASE   0x000A0000

5.5.2 Specification of OHCI Endpoint Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI endpoint data structure. The OHCI endpoint data structure must be able to accommodate the number of endpoint pipes used for control transfers, bulk transfers, interrupt transfers, and isochronous transfers. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 16
   #define USB_OHCI_NUM_ENDPOINT 16

5.5.3 Specification of OHCI Endpoint Descriptor Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI endpoint descriptor data structure. The OHCI endpoint descriptor data structure must be able to accommodate the number of endpoint pipes used for control transfers, bulk transfers, interrupt transfers, and isochronous transfers, plus 31 more for interrupt transfer scheduling. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 64
   #define USB_OHCI_NUM_ED   64

5.5.4 Specification of OHCI Transfer Descriptor Data Structure Maximum Memory Size

Specify the maximum memory size for the OHCI transfer descriptor data structure. The OHCI transfer descriptor data structure is used as a transfer management descriptor in control transfers, bulk transfers, interrupt transfers, and isochronous transfers. The maximum data size that can be transferred with a single transfer descriptor is 8,192 bytes. In addition, control transfers require one qTD for each Setup stage, Data stage, and Status stage. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 256
   #define USB_OHCI_NUM_TD   256

5.5.5 Specification of OHCI Maximum Isochronous Device Count

Specify the maximum number of OHCI isochronous devices. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 4
   #define USB_OHCI_ISO_MAXDEVICE   4

5.5.6   Specification of OHCI Maximum Isochronous Data Transfer Size

Specify the maximum data transfer size for OHCI isochronous devices. The transfer size value specifies the maximum transfer size for a single OHCI isochronous transfer (one transaction). The largest supported maximum data size setting is 1023. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 256
            #define USB_OHCI_ISO_MAX_PACKET_SIZE   256

5.5.7   Specification of OHCI Maximum Isochronous frame count

Specify the maximum frame count for OHCI isochronous devices. The frame count value specifies the maximum number of frames for isochronous transfer . The setting value must be power of 2 and the largest supported maximum frame count is 8. Set a value that matches the characteristics of the system under development.

Example: Setting a value of 8
            #define USB_OHCI_ISO_MAX_FRAME   8

5.5.8   Specification of OHCI Timeout Duration

Specify the OHCI timeout duration in msec units.

Example: Setting a value of 3000
            #define USB_OHCI_TIMEOUT   3000

6. Sample Program

6.1 Overview

The method of preparing HDCD to work with USB-BASIC-F/W is described below.

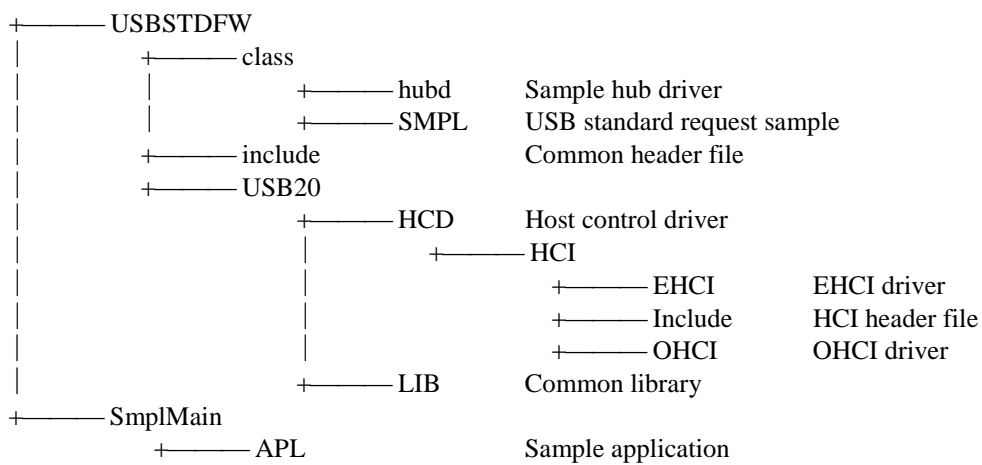The USB-BASIC-F/W sample program comprises the following project.

- Itron_StdFw: μITRON version

6.2 List of Files

6.2.1 Folder Structure

The folder structure of the files supplied with USB-BASIC-F/W is shown below.

< HEW workspace: USBSTDFW >

```
+――――― USBSTDFW
│        +――――― class
│        │        +――――― hubd        Sample hub driver
│        │        +――――― SMPL        USB standard request sample
│        +――――― include              Common header file
│        +――――― USB20
│                 +――――― HCD         Host control driver
│                 │       +――――― HCI
│                 │                +――――― EHCI        EHCI driver
│                 │                +――――― Include     HCI header file
│                 │                +――――― OHCI        OHCI driver
│                 +――――― LIB         Common library
+―――――SmplMain
          +――――― APL                 Sample application
```

6.2.2   List of Files

The files supplied with USB-BASIC-F/W are listed below.

Table 6.1   List of Files

Folder	File Name	Description
HCD	r_usb_hControlRW.c	Control read/write process
HCD	r_usb_hDriver.c	HCD task
HCD	r_usb_hDriverAPI.c	HCD/MGR API functions
HCD	r_usb_hManager.c	MGR task
HCD	r_usb_hSignal.c	USB signal control, oscillation control process
HCD	r_usb_hStdFunction.c	USB functionality extended library function
HCD	r_usb_hLibUSBIP.c	USB library function
HCI	r_usb_hHci.c	EHCI/OHCI functionality provision function
HCI\EHCI	r_usb_hEhciMain.c	EHCI main function
HCI\EHCI	r_usb_hEhciMemory.c	EHCI memory resource function
HCI\EHCI	r_usb_hEhciTransfer.c	EHCI transfer process
HCI\include	r_usb_hEhciDefUsr.h	EHCI user setting definitions
HCI\include	r_usb_hEhciExtern.h	EHCI external reference definitions
HCI\include	r_usb_hEhciTypedef.h	EHCI variable definitions
HCI\include	r_usb_hHciLocal.h	HCI common reference header file
HCI\include	r_usb_hOhciDefUsr.h	OHCI user setting definitions
HCI\include	r_usb_hOhciExtern.h	OHCI external reference definitions
HCI\include	r_usb_hOhciTypedef.h	OHCI variable definitions
HCI\OHCI	r_usb_hOhciMain.c	OHCI main function
HCI\OHCI	r_usb_hOhciMemory.c	OHCI memory resource function
HCI\OHCI	r_usb_hOhciTransfer.c	OHCI transfer process
LIB	r_usb_cDataIO.c	Data read/write, FIFO access process
LIB	r_usb_cIntHandler.c	USB interrupt handler
LIB	r_usb_cLibUSBIP.c	USB library function
include	r_usb_cDefUSBIP.h	USB driver definitions
include	r_usb_cItron.h	System header file
include	r_usb_cMacItron.h	Macro definitions (scheduler macro)
include	r_usb_cMacPrint.h	Macro definitions (debug display macro)
include	r_usb_cTypedef.h	Variable type definitions
include	r_usb_cDefHCIIP.h	59xIP compatibility definitions
include	r_usb_cDefUsr.h	User setting (hardware operation specification) definitions
include	r_usb_cDefUsrPb.h	USB driver definitions
include	r_usb_cExtern.h	USB-BASIC-F/W external reference definitions
include	r_usb_cRevision.h	Common library revision specification
include	r_usb_cKernelId.h	Macro definitions(Identifiers for μ itronOS)
include	r_usb_hHci.h	External reference definitions for provision of EHCI/OHCI functionality
class\hubd	r_usb_hHubsys.c	HUBCD function
class\SMPL	r_usb_smp_cSub.c	Common library function
calss\SMPL	r_usb_smp_hSub.c	Host standard request
SmplMain	main.c	Sample main program
SmplMain	SH7786.c	Process for SH7786
SmplMain	SH7786_Extern.h	External reference definitions for SH7786
APL	r_usb_HSMPL_apl.c	Host port 1 sample application
APL	r_usb_cdata.c	Data transfer data for port 1 sample application
APL	r_usb_cdata.h	Data transfer settings for port 1 sample application

6.3   μITRON System Resources

The μITRON resources used by USB-BASIC-F/W are listed below. They are defined in the **r_usb_cKernelId.h** header file.
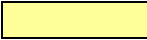
Table 6.2   μITRON Resources

	Name	Description
Task stack size: USB_TSK_STK(0x0800)	usb_hstd_HcdTask	HCD task Task_ID: USB_HCD_TSK Task priority: USB_HCD_PRI
	usb_hstd_MgrTask	MGR task Task_ID: USB_MGR_TSK Task priority: USB_MGR_PRI
	Main_Task	Main task Task_ID: USB_SMP_TSK Task priority: USB_SMP_PRI
	usb_hHubTask	HUBCD task Task_ID: USB_HUB_TSK Task priority: USB_HUB_PRI
	HCD System Task	HCD SYS task Task_ID: USB_HCI_SYS_TSK Task priority: USB_HCI_SYS_PRI
	HCD Transfer Task	HCD TRN task Task_ID: USB_HCI_TRN_TSK Task priority: USB_HCI_TRN_PRI
Mailbox max. priority: 1 Waiting task queue: FIFO order Message queue: FIFO order	USB_HCD_MBX	HCD mailbox ID
	USB_MGR_MBX	MGR mailbox ID
	USB_CLS_MBX	HDCD mailbox ID
	USB_HUB_MBX	HUBCD mailbox ID
	USB_HUBC_MBX	Mailbox ID for HUBCD control transfers
Fixed-length memory pool block count: USB_BLK_CNT(0x20) Block size: USB_BLK_SIZ(0x40) Waiting task queue: FIFO order	USB_HCD_MPL	HCD memory pool ID
	USB_MGR_MPL	MGR memory pool ID
	USB_CLS_MPL	HDCD memory pool ID
	USB_HUB_MPL	HUBCD memory pool ID
Semaphore Initial value: 1 Maximum number: 1 Wait task queue: FIFO order	ID: USB_TRN_SEM	Semaphore for transfer management
Semaphore Initial value: 0 Maximum number: 32 Wait task queue: FIFO order	ID: USB_HCI_SYS_SEM	Semaphore for HCD system
	ID: USB_HCI_TRN_SEM	Semaphore for HCD transfer
Semaphore Initial value: 1 Maximum number: 1 Wait task queue: FIFO order	ID: USB_HCI_MEM_SEM	Semaphore for HCI memory management
Semaphore Initial value: 0 Maximum number: 1 Wait task queue: FIFO order	ID: USB_HCI_DC_SEM	Semaphore for HCI device detach notification

	Name	Description
Semaphore Initial value: 0 Maximum number: 1 Wait task queue: FIFO order	ID: USB_HCI_TRCC_S_SEM	Transfer cancel end notification semaphore for HCD system task
Semaphore Initial value: 0 Maximum number: 1 Wait task queue: FIFO order	ID: USB_HCI_TRCC_T_SEM	Transfer cancel end notification semaphore for HCD transfer task
OS base timer	Hardware timer	1ms

6.4   Sections

The sections used by USB-BASIC-F/W are described bellow. .

Section Name	Description
P_usblib	USB firmware common processing program area
P_hcd	HCD, HCI program area
P_hub	HUB class program area
C_usblib	USB firmware common processing fixed-value data area
C_hcd	HCD, HCI fixed-value data area
C_hub	HUB class fixed-value data area
D_usblib	USB firmware common processing initialized memory area
D_hcd	HCD, HCI initialized memory area
D_hub	HUB class initialized memory area
B_usblib	USB firmware common processing uninitialized memory area
B_hcd	HCD, HCI uninitialized memory area
B_hub	HUB class uninitialized memory area
B_ehci_non_cache	EHCI transfer uninitialized memory area[1] [4]
B_ohci_non_cache	OHCI transfer uninitialized memory area[2]
B_hcd_non_cache	Control transfer uninitialized memory area[3]

[yellow]	: Program area
[pink]	: Fixed data area
[cyan]	: Initialized data area
[green]	: Uninitialized data area

Notes: 1. Assign B_ehci_non_cache to an address aligned at a 4 KB boundary in a non-cached area.
      2. Assign B_ohci_non_cache to an address aligned at a 256 bytes boundary in a non-cached area.
      3. Assign to a non-cached area.
      4. Assign isochronous transfer descriptor "ehci_Itd[USB_EHCI_NUM_ITD]" to an address aligned at a 64byte boundary in B_ehci_non_cache section.

6.5   μITRON configurator

USB-BASIC-F/W uses the ITRON configurator to make the following settings.

1. Kernel operating condition: Kernel interrupt mask level **14**
2. Time management function: Use time management function.
   Timer interrupt number **0x0400**
   Timer interrupt level **13**
   Timer event handler stack size **0x0200**
   Time tick cycle **1 ms**
3. Service call selection: Use all service calls.
4. Interrupts, CPU exception handlers
   Power On Reset
   TRAPA
   SYSTEM TIMER
   UsbInt_Hand: C language
   _kernel_tmrint: C language
5. Initialization routine **PowerON_Reset_PC : stack size 0x0100 : C language**

6.6   Creating and Starting μITRON Tasks

The initialization routine (PowerON_Reset() function) of USB-BASIC-F/W creates the main task, memory pools, and mailboxes and then starts the main task. The main task creates and starts the HCD and MGR tasks. The HDCD task is created and started according to the Set_Configuration request reply timing.

6.7   µITRON System Calls

USB-BASIC-F/W uses the following system calls.

Table 6.3   System Calls

System Call	Description
USB_CRE_TSK	Creates a task.
USB_DEL_TSK	Deletes a task.
USB_STA_TSK	Starts a task.
USB_TER_TSK	Terminates a task.
USB_DLY_TSK	Waits for a specified duration.
USB_CRE_MBX	Creates a mailbox.
USB_DEL_MBX	Deletes a mailbox.
USB_SND_MSG	Transmits a message from a task (interrupt) to a task.
USB_ISND_MSG	Transmits a message from a task (interrupt) to a task.
USB_RCV_MSG	Receives a message from the message buffer.
USB_PRCV_MSG	Receives a message from the message buffer.
USB_TRCV_MSG	Receives a message with a timeout.
USB_CRE_MPL	Creates a memory pool.
USB_DEL_MPL	Deletes a memory pool.
USB_PGET_BLK/ USB_IPGET_BLK	Gets a variable-length memory block when triggered by a task (interrupt).
USB_REL_BLK	Releases a variable-length memory block.
USB_CRE_SEM	Creates a semaphore.
USB_WAI_SEM	Gets a semaphore.
USB_POL_SEM	Checks a semaphore.
USB_SIG_SEM/ USB_ISIG_SEM	Releases a semaphore when triggered by a task (interrupt).
USB_DEL_SEM	Deletes a semaphore.

7. Host Driver (HCD)

7.1 Basic Functionality

HCD interprets requests to the hardware from MGR, HUBCD, and HDCD, and issues corresponding data transfer or hardware control requests to HCD TRN and HCD SYS. HCD provides the following functionality.

(1) Handling control transfer request
(2) Handling data transfer (bulk/isochronous/interrupt) requests
(3) Handling USB bus reset requests and notification of reset handshake results
(4) Issuing of suspend and resume signals

7.2 Issuing Requests to HCD

API functions, described below, are used to issue hardware control requests to HCD. Note that bus state update requests for connected devices can either be controlled directly by HCD or controlled via the USB hub. MGR determines the device address and issues a control request to the HCD or HUBCD task.

7.3 Notes on Using HCD

Bear in mind the following items when performing USB communication by using HCD.

7.3.1 Bus Occupation Rate and Pipe Contention

HCD can issue communication requests to multiple devices. However, HCD does not calculate the USB bus occupation rate or check for contention among the communication pipes used by HDCD. To prevent pipe contention when multiple instances of HDCD are running, it is necessary to make modifications to HDCD. Note that HUBCD uses pipes 6 to 10.

7.3.2 Device Addresses

MGR enumerates connected devices using the user-specified value of USB_DEVICEADDR. HUBCD automatically assigns device addresses to devices connected to down ports, starting with USB_DEVICEADDR + 1.

7.3.3 Running Multiple Instances of HDCD

MGR cannot perform enumeration for multiple devices simultaneously.

7.3.4 Starting HDCD

When the device and configuration are established (SET_CONFIGURATION request response), MGR (HUBCD) notifies HDCD of the configuration by using the registered callback function. HDCD when enables data communication for transactions starting with data reception.

7.4 HCD Task Startup Sequence

The startup sequence for the HCD and MGR tasks is shown below.
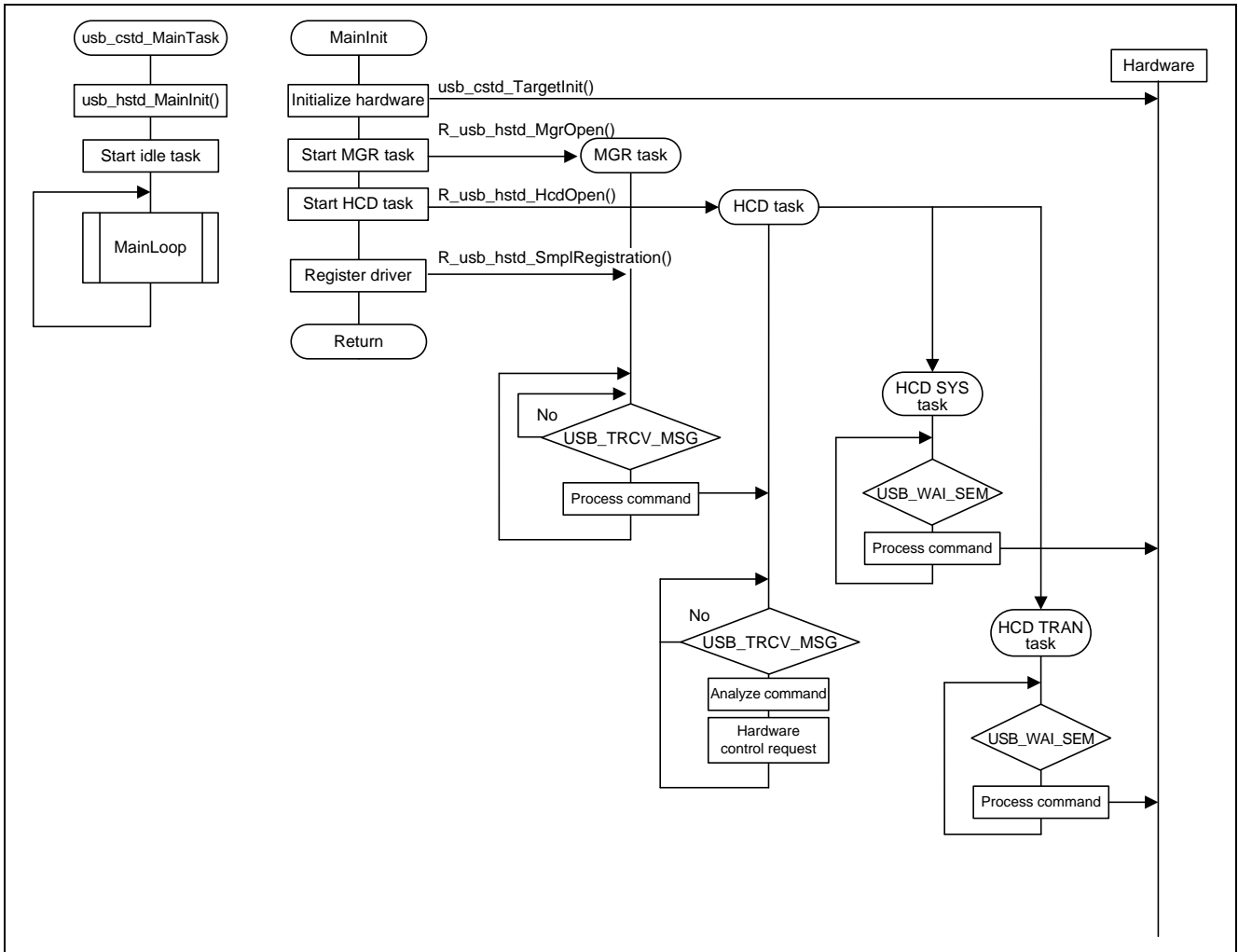
Figure 7.1 HCD/MGR Startup Sequence

7.5   HCD Outline Flowchart

An outline flowchart of HCD is shown below.
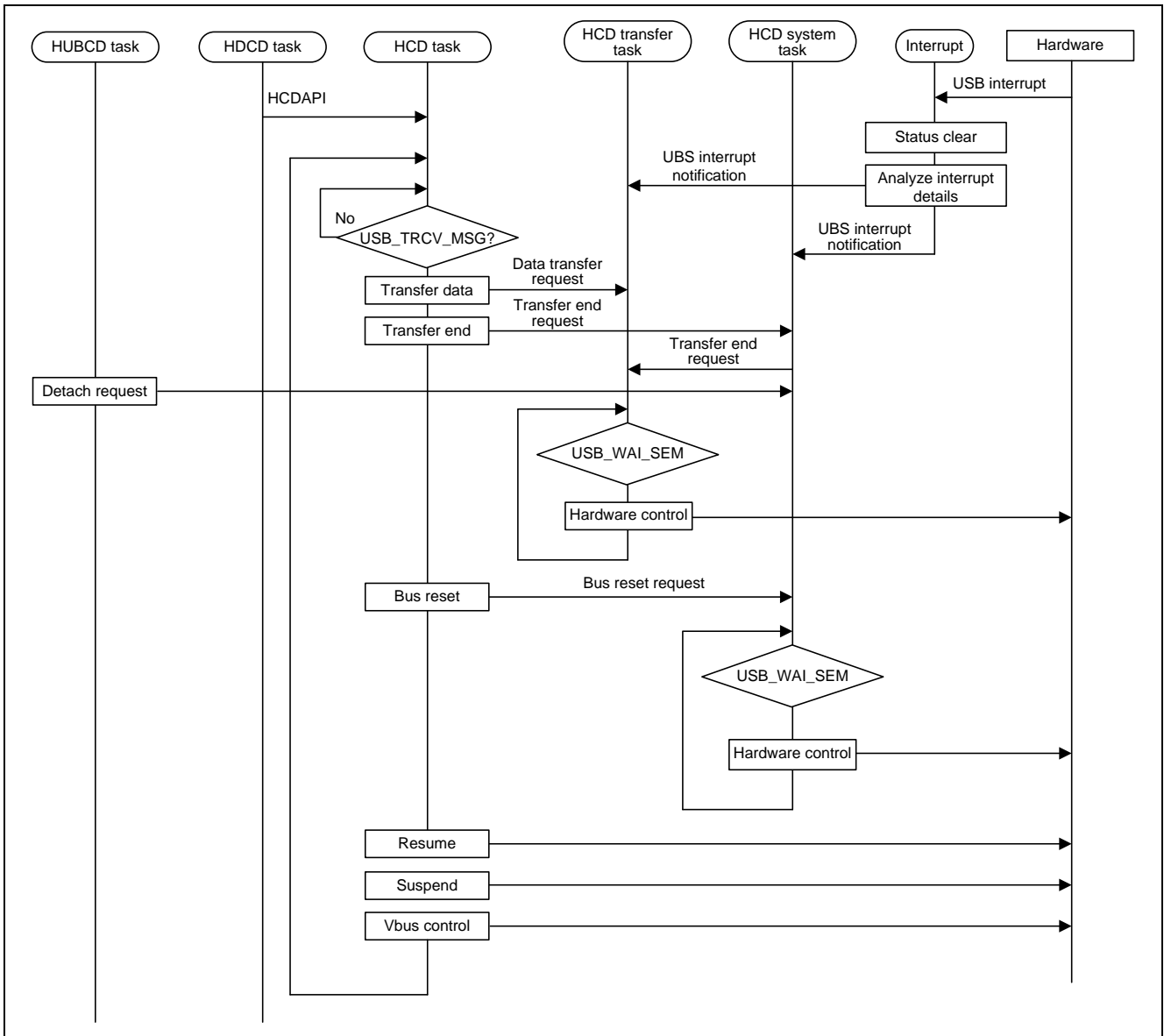
Figure 7.2   HCD Outline Flowchart

7.6 HCD API Functions

MGR, HUBCD, and HDCD issue hardware control requests by using API functions. The API functions are contained in the file **r_usb_hDriverAPI.c**, and the API function return values are scheduler macro error codes.

Table 7.1   List of HCD API Functions

	Function Name	Description
1	R_usb_hstd_HcdOpen()	Start HCD task
2	R_usb_hstd_HcdClose()	End HCD task
3	R_usb_hstd_TransferStart()	Data transfer request
4	R_usb_hstd_SetPipeRegistration()	Pipe configuration setting request
5	R_usb_hstd_TransferEnd()	Data transfer forced end request
6	R_usb_hstd_ChangeDeviceState()	USB device state change request

7.6.1   Details of HCD API Functions

Table 7.2   R_usb_hstd_HcdOpen()

Name	Start HCD Task		
Call format	USB_ER_t R_usb_hstd_HcdOpen(void)		
Arguments	void	—	—
Return values	USB_ER_t	—	Error code
Description	This function starts the HCD task. It creates the HCD Transfer and HCD System tasks; makes initial global variable settings; creates tasks, mailboxes, and memory pools; and starts the tasks. The HCD task then waits for requests from the hardware, MGR, and HDCD.		
Notes	Calling this function starts the HCD task. Do not call it again after the task has started.		

Table 7.3   R_usb_hstd_HcdClose()

Name	End HCD Task		
Call format	USB_ER_t R_usb_hstd_HcdClose(void)		
Arguments	void	—	—
Return values	USB_ER_t	—	Error code
Description	This function ends the HCD task. It releases the mailboxes and memory pools, and terminates the task.		
Notes	Calling this function ends the HCD task. Do not call it again after the task has closed.		

Table 7.4   R_usb_hstd_TransferStart()

Name	Data Transfer Request		
Call format	USB_ER_t R_usb_hstd_TransferStart(USB_UTR_t *utr_table)		
Arguments	USB_UTR_t	*utr_table	Transfer information. See structure definitions.
Return values	USB_ER_t		Error code
Description	This function performs data transfer via the pipes. When data transfer ends (specified data size reached, short packet received, or error occurred), the callback function argument (utr_table) is used to send notification of the remaining transmit/receive data length, status, error count, and transfer end. The data transfer information indicated in utr_table is transmitted to HCD as a message, and HCD performs the appropriate processing.		
Notes	The members of the structure indicated in the argument include pipe number, transfer data start address, transfer data length, setup packet address, status, callback function at end, and error count.		

Table 7.5   R_usb_hstd_SetPipeRegistration()

Name	Pipe Configuration		
Call format	USB_ER_t R_usb_hstd_SetPipeRegistration (uint16_t *table, uint16_t pipe)		
Arguments	uint16_t	*table	Pipe information table
	uint16_t	pipe	Pipe number
Return values	USB_ER_t		Error code
Description	This function specifies the pipe configuration information. The settings for the pipes are based on the contents of the information table (devicedriver.pipetable) registered at HDCD registration. At processing end, notification that the settings are complete is sent by using the usb_cstd_ClassProcessResult callback function. This information is transmitted to HCD as a message, and HCD performs the appropriate processing.		
Notes	The contents of the information table (devicedriver.pipetable) should be specified by the user after analyzing the descriptor details, etc. This function was originally for making hardware pipe configuration settings required by the R8A6659x ASSP/USB IP. Since the Basic Firmware with EHCI support uses some pipe configuration information items, this function is required for making settings.		

Table 7.6   R_usb_hstd_TransferEnd()

Name	Data Transfer Forced End Request		
Call format	void R_usb_hstd_TransferEnd(uint16_t pipe, uint16_t status)		
Arguments	uint16_t	pipe	Pipe number
	uint16_t	status	
Return values	USB_ER_t		Error code
Description	This function forcibly ends data transfer via the pipes. When a data transfer forced end occurs, the R_usb_hstd_TransferStart() function's callback function argument (utr_table) is used to send notification of the remaining transmit/receive data length, status, error count, and forced transfer end. This information is transmitted to HCD as a message, and HCD performs the appropriate processing.		
Notes	This function executes the same callback function as when R_usb_hstd_TransferStart() is executed.		

Table 7.7   R_usb_hstd_ChangeDeviceState()

Name	Device State Change Request		
Call format	USB_ER_t R_usb_hstd_ChangeDeviceState(USB_CB_INFO_t complete, uint16_t msginfo, uint16_t rootport)		
Arguments	USB_CB_INFO_t	complete	Callback function
	uint16_t	msginfo	Message information
		connect_inf	Connection state
	uint16_t	rootport	Port number
Return values	USB_ER_t		Error code
Description	This function manages the device state. At processing end, notification that the settings are complete is sent by using a callback function. This information is transmitted to HCD as a message, and HCD performs the appropriate processing.		
Notes	The message information used for state management is as follows. #define USB_MSG_HCD_ATTACH: Request for transition to connected state #define USB_MSG_HCD_DETACH: Request for transition from connected to disconnected state #define USB_MSG_HCD_USBRESET: Request to issue USB reset #define USB_MSG_HCD_SUSPEND: Request for transition to suspended state #define USB_MSG_HCD_RESUME: Request to issue resume signal #define USB_MSG_HCD_REMOTE: Request for transition to suspended with remote wakeup state #define USB_MSG_HCD_VBON: Vbus output start request #define USB_MSG_HCD_VBOFF: Vbus output end request #define USB_MSG_HCD_CLR_STALL: Stall clear request		

7.7 HCD Callback Functions

Table 7.8 R_usb_hstd_TransferStart Callback

Name	Data Transfer Request Callback Function		
Call format	(*USB_CB_INFO_t)(USB_UTR_t*);		
Arguments	USB_UTR_t*		USB_UTR_t pointer argument of R_usb_hstd_TransferStart() function
Return values	—	—	—
Description	This function is executed at data transfer end (when data transfer of the size specified by the application completes or transfer ends because a short packet is received, etc.). It updates the remaining transmit/receive data length and the error count.		
Notes			

Table 7.9 R_usb_hstd_ChangeDeviceState(USB_MSG_HCD_USBRESET) Callback

Name	USB Bus Reset Callback Function		
Call format	(*USB_CB_INFO_t)(uint16_t,uint16_t)		
Arguments	uint16_t		USB_NOCONNECT: Not connected USB_HSCONNECT: High-speed device USB_FSCONNECT: Full-speed device USB_LSCONNECT: Low-speed device
	uint16_t		NOARGUMENT: Not used
Return values	—	—	—
Description	This function is executed when USB bus reset processing finishes. It returns the communication speed of the connected device as an argument. The not connected argument is returned when detach detection occurs while USB bus reset is in progress or the speed is undefined.		
Notes			

Table 7.10 Other Callbacks

Name	Other Callback Functions		
Call format	(*USB_CB_INFO_t)(uint16_t,uint16_t)		
Arguments	uint16_t		NOARGUMENT: Not used
	uint16_t		msginfo: Command classification
Return values	—	—	—
Description	USB_MSG_HCD_ATTACH: Executed at end of attach processing. USB_MSG_HCD_DETACH: Executed at end of detach processing. USB_MSG_HCD_SUSPEND: Executed at end of suspend processing. USB_MSG_HCD_RESUME: Executed at end of resume processing. USB_MSG_HCD_REMOTE: Executed at end of remote wakeup processing. USB_MSG_HCD_VBON: Executed at end of Vbus on processing. USB_MSG_HCD_VBOFF: Executed at end of Vbus off processing. USB_MSG_HCD_SETDEVICEINFO: Executed at end of pipe setting processing. usb_hstd_ControlEnd: Executed at end of data transfer. USB_MSG_HCD_CLRSEQBIT: Executed when sequence toggle bit is cleared to 0. USB_MSG_HCD_SETSEQBIT: Executed when sequence toggle bit is set to 1.		
Notes			

7.8   Structure Definitions

7.8.1   USB_HCDINFO_t Structure

The message structure transmitted to HCD by the R_usb_hstd_ChangDeviceState(), R_usb_hstd_SetPipeRegistration(), R_usb_hstd_TransferEnd(), and R_usb_hhub_ChangeState() functions is described below.

```
typedef struct {
  USB_MH_t        msghead;  // Message heater used by the OS
  uint16_t        msgInfo;  // Message information used by USB-BASIC-F/W
  uint16_t        keyword;  // Sub-information (port number, pipe number, etc.)
  void            *tranadr;// Specifies the pipe information table for R_usb_hstd_SetPipeRegistration() only.
  USB_CB_INFO_t   complete;// Callback function at processing end
}USB_HCDINFO_t;
```

Table 7.11   Members of USB_HCDINFO_t Structure

Variable	Description
msghead	This message header is used by the OS, so the client should not make use of it.
msgInfo	This message header is used by USB-BASIC-F/W, so the client should not make use of it.
keyword	This differs according to the message, as follows. Pipe number: R_usb_hstd_ChangDeviceState(), R_usb_hstd_SetPipeRegistration(), R_usb_hstd_TransferEnd() Device address: R_usb_hhub_ChangeState()
*tranadr	R_usb_hstd_SetPipeRegistration() function: Specifies the address of the pipe setting information table. Other API functions: Ignored even if specified.
complete	Specifies the address of the function to be executed when HCD completed its current processing. Use the type declaration **void (*USB_CB_INFO_t)(uint16_t,uint16_t)** for the callback function. For USB reset signal output control by the R_usb_hstd_ChangeDeviceState() function, the reset handshake result is returned as the first argument of the callback. For other functions, the callback indicates that processing has completed.

7.8.2   Information Registered in HCD (USB_HCDREG_t Structure)

The structure for registering HDCD is described below. See section 11 for the timing of callback function execution.

```
typedef struct {
    uint16_t        rootport;       /* root port */
    uint16_t        devaddr;        /* Device address */
    uint16_t        devstate;       /* Device state */
    uint16_t        ifclass;        /* Interface class */
    uint16_t        *tpl;           /* Target peripheral list */
    uint16_t        *pipetbl ;      /* Pipe define table address */
    USB_CB_INFO_t   classinit;      /* Driver init */
    USB_CB_CHECK_t  classcheck;     /* Driver check */
    USB_CB_INFO_t   devconfig;      /* Device configured */
    USB_CB_INFO_t   devdetach;      /* Device detach */
    USB_CB_INFO_t   devsuspend;     /* Device suspend */
    USB_CB_INFO_t   devresume;      /* Device resume */
    USB_CB_INFO_t   overcurrent;    /* Device over current */
}USB_HCDREG_t;
```

Table 7.12   Members of USB_HCDREG_t Structure

Variable	Description
rootport	Used by HCD. Registers the connected port number.
devaddr	Used by HCD. Registers the device address.
devstate	Used by HCD. Registers the device connection state.
ifclass	Register the class code of the interface operated by HDCD.
*tpl	Register a list of the target peripherals operated by HDCD.
*pipetbl	Register the address of the pipe information table.
classinit	Register the function to be started at driver registration.
classcheck	Register the function to be started at HDCD checking. It is called when TPL matches.
devconfig	Register the function to be started at transition to the configuration state. It is called when a SET_CONFIGURATION request completes.
devdetach	Register the function to be started at transition to the detached state.
devsuspend	Register the function to be started at transition to the suspended state.
devresume	Register the function to be started at transition to the resume state.
overcurrent	Register the function to be started at overcurrent detection.

A separate TPL is created for each device class. In a TPL the header ID and product ID are registered as a set. Register all supported sets in the relevant device class.

To register is device as a device class that supports all devices, register USB_NOVENDOR and USB_NOPRODUCT as a set.

```
const uint16_t tpl[] = {
    1,                      /* Number of lists */
    0,                      /* Reserved */
    USB_NOVENDOR,           /* Vendor ID */
    USB_NOPRODUCT,          /* Product ID */
};
```

8.   HCD Transfer (HCD TRN)

8.1   Basic Functionality

The HCD Transfer program controls data transfers. It performs data transfers according to data transfer requests from HCD. HDCD, etc., are notified of the control results by using callback functions. HCD Transfer provides the following functionality.

(1) Control transfer and result notification
(2) Data transfer and result notification

8.2   Issuing Requests to HCD Transfer

Data transfer requests are issued to HCD Transfer by HCD. After HUBCD or HDCD sends a transfer request and makes transfer information settings to HCD via the API, HCD issues a transfer request to HCD TRN.

A direct notification of the transfer result is sent to the request source by using a callback function.

8.3   HCD Transfer Task Startup Sequence

See figure 7.1, HCD/MGR Startup Sequence, for the startup sequence of the HCD Transfer task.

8.4   HCD Transfer Outline flowchart

See figure 7.2, HCD Outline Flowchart, for an outline flowchart of the HCD Transfer task.

9. HCD System (HCD SYS)

9.1 Basic Functionality

The HCD System program performs port state management and hardware control. It performs hardware control according to hardware control requests from HCD and HUBCD, and it sends notifications to MGR when port state changes are detected.

HCD System provides the following functionality.

(1) Port state change (attach/detach/overcurrent) notification
(2) Bus reset hardware control
(3) Detach processing hardware control

9.2 Port State Change Notification

HCD System is notified by the USB interrupt handler when the port state changes. After receiving a port state change notification, HCD System checks the port state and notifies MGR if it is attached, detached, or overflow by using a callback function.

9.3 Bus Reset Hardware Control

Requests for bus reset hardware control are sent to HCD System by HCD as messages. After receiving such a request, HCD System performs bus reset control accordingly.

9.4 Detach Processing Hardware Control

Requests for detach processing hardware control are sent to HCD System by HUBCD as messages. After receiving such a request, HCD System performs detach processing control accordingly.

9.5 HCD System Task Startup Sequence

See figure 7.1, HCD/MGR Startup Sequence, for the startup sequence of the HCD Transfer task.

9.6 HCD System Outline Flowchart

See figure 7.2, HCD Outline Flowchart, for an outline flowchart of the HCD System task.

10. Host Control Transfer

10.1   Overview

A control transfer can be performed by notifying HCD of the USB request setup packet and data transmit/receive user buffer (see 13.2, USB Communication Structure).

10.1.1   Basic Specifications

When HCD receives a data transfer request, it checks the contents of the request and makes the settings necessary for the transfer, after which it sends a transfer request to HCD TRN. After receiving the request, HCD TRN performs control transfer scheduling and executes a control transfer, using the setup information and user buffer specified in the structure as transfer memory.

Ensure that the user buffer is sufficient to accommodate the size of the data transmitted or received in the Data stage.

10.1.2   Notification of Transfer Result

When the control transfer ends, HCD TRN executes the callback function specified at submit to notify HDCD of control transfer end.

The communication result notification sent by HCD TRN to HDCD can be any one of the following.

- USB_CTRL_END: Successful control transfer end
- USB_DATA_STALL: STALL response or MAXP error in Data stage or Status stage
- USB_DATA_OVR: Receive data size over in Data stage
- USB_DATA_ERR: No response detected
- USB_DATA_STOP: Forced end of control transfer (including end when detach detected)

10.2　Control Transfer Operation

10.2.1　Control Write Operation

HCD TRN uses the following procedure to perform a data control transfer.

[1] HCD TRN starts a control transfer upon receipt of a transfer request from HCD.

[2] After creating a transfer schedule based on the transfer request notification from HCD, HCD TRN performs hardware control and starts the transfer. (Stage management is implemented by the hardware according to the transfer schedule.)

[3] After transfer end, the interrupt handler notifies HCD Transfer that a transfer end interrupt was generated.

[4] After receiving the transfer end notification, HCD TRN sends notification of transfer end via the callback function specified by HCD.
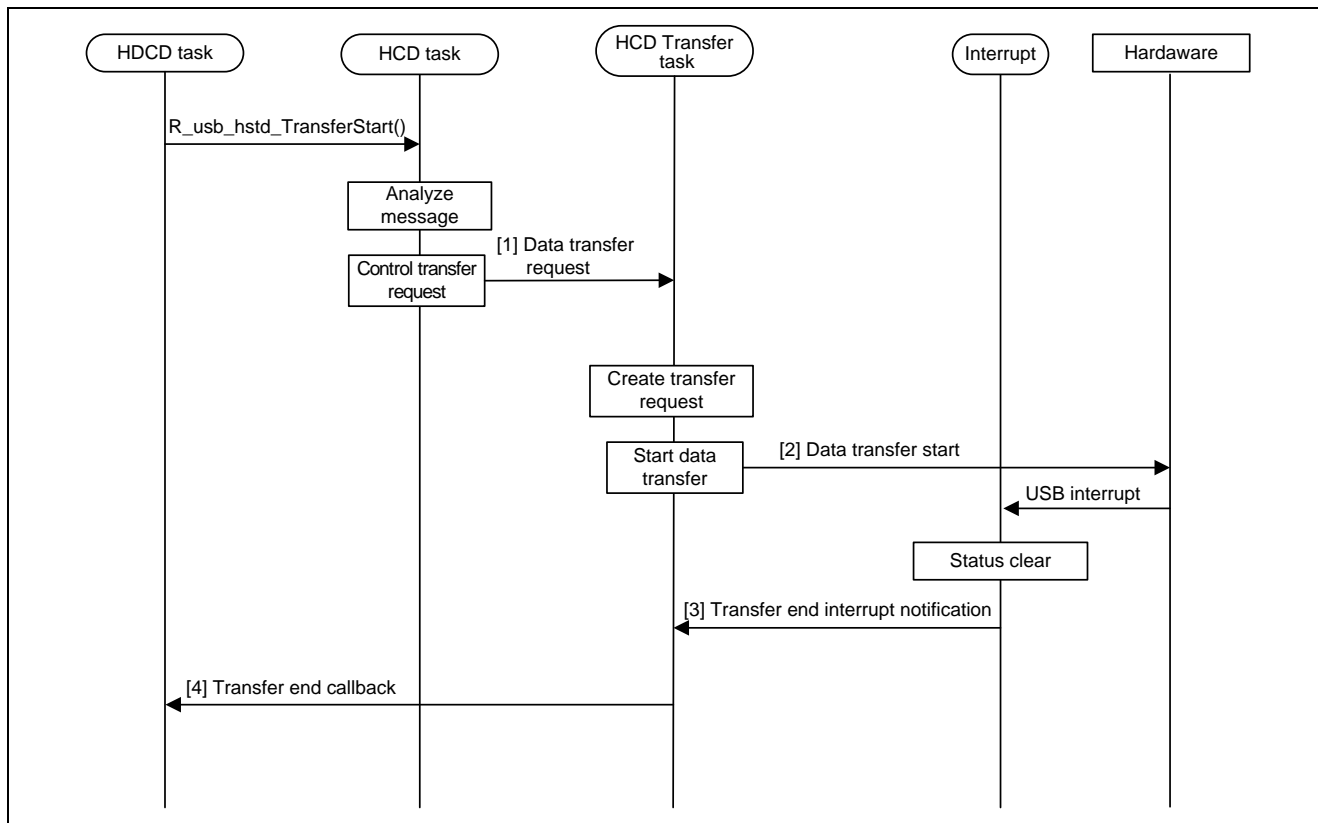
Figure 10.1　Control Write Outline Flowchart

10.2.2   Control Read Operation

HCD TRN uses the following procedure to perform a control read transfer.

[1] HCD TRN starts a control transfer upon receipt of a transfer request from HCD.

[2] After creating a transfer request based on the transfer request notification from HCD, HCD TRN makes hardware settings and starts the transfer. (Stage management is implemented by the hardware according to the transfer schedule.)

[3] After transfer end, the interrupt handler notifies HCD Transfer that an interrupt was generated.

[4] After receiving the transfer end notification, HCD TRN copies the receive data to the user buffer.

[5] HCD TRN sends notification of transfer end via the callback function specified by HCD.
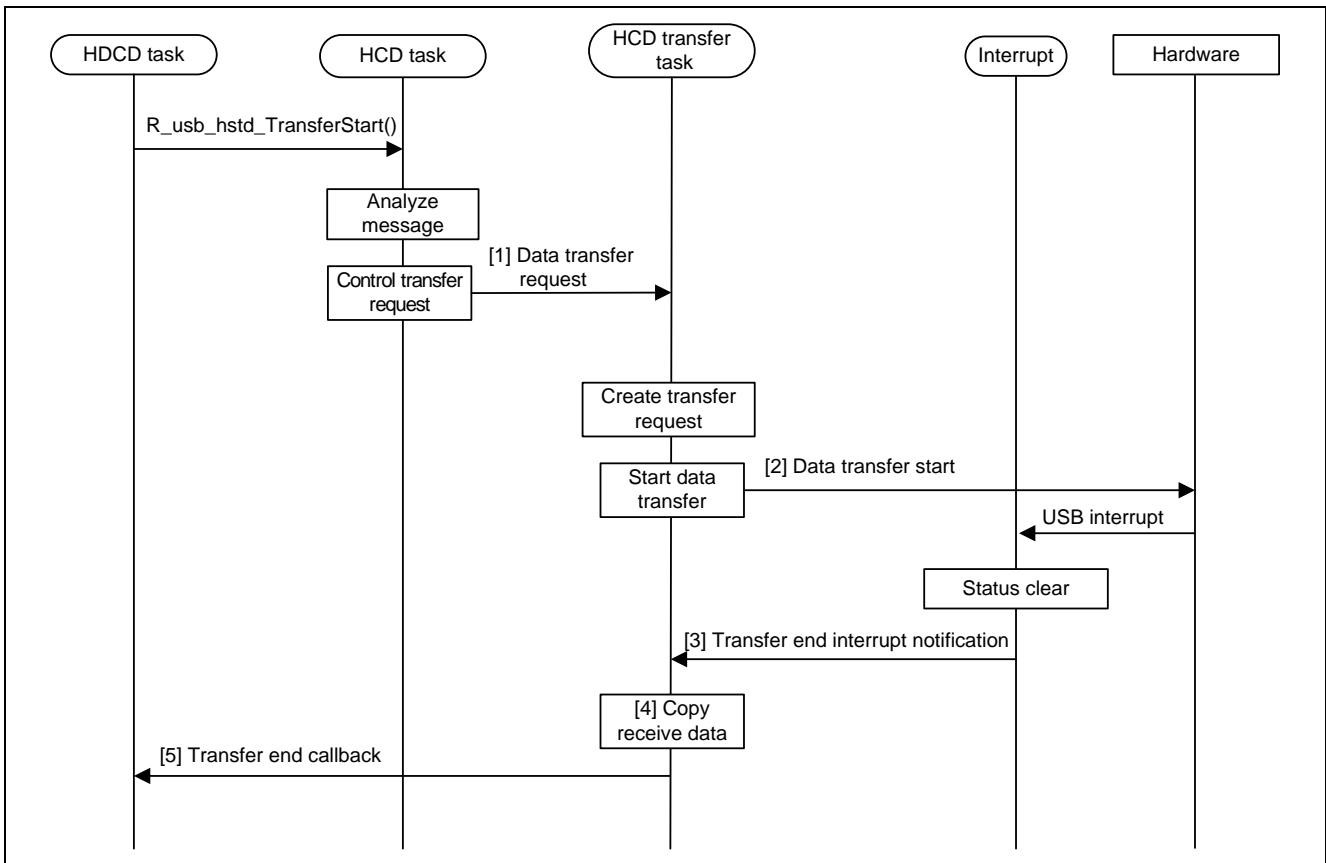
Figure 10.2   Control Read Outline Flowchart

10.3   Control Transfer Sequence
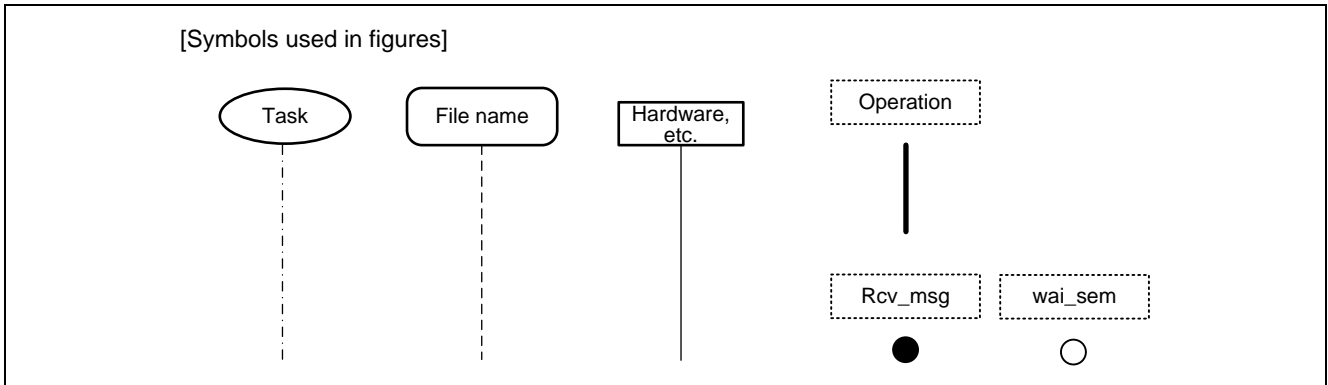
10.3.1   No-Data Control Sequence
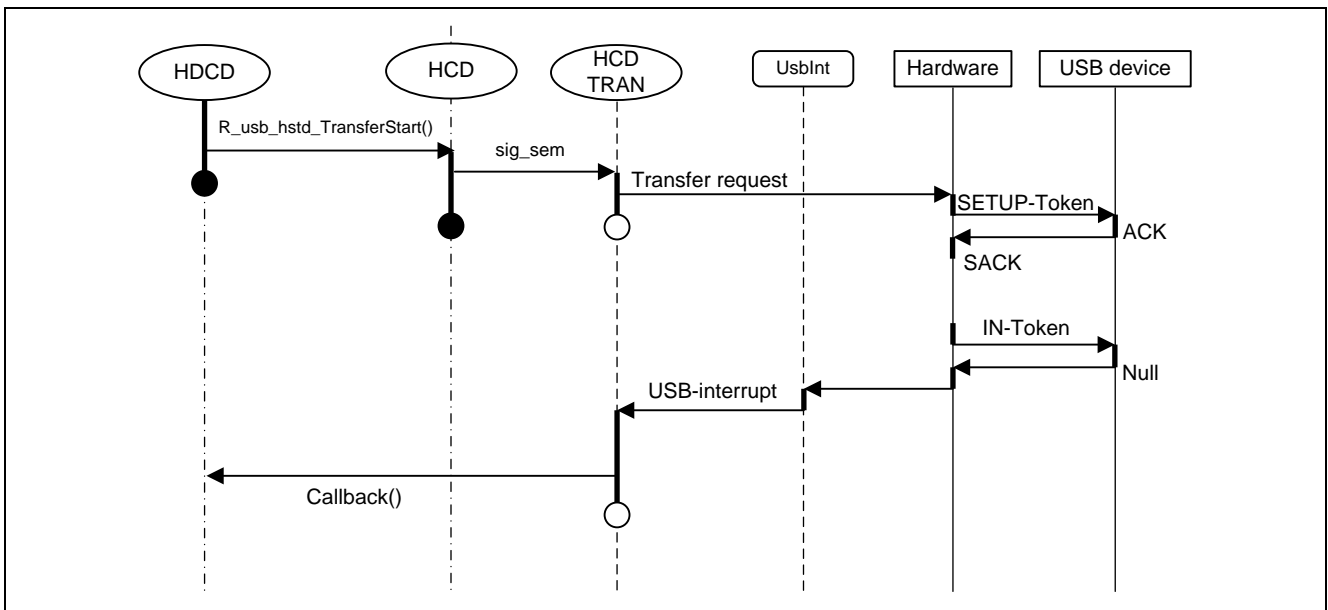
Figure 10.3   No-Data Control Sequence

10.3.2   Control Write Sequence
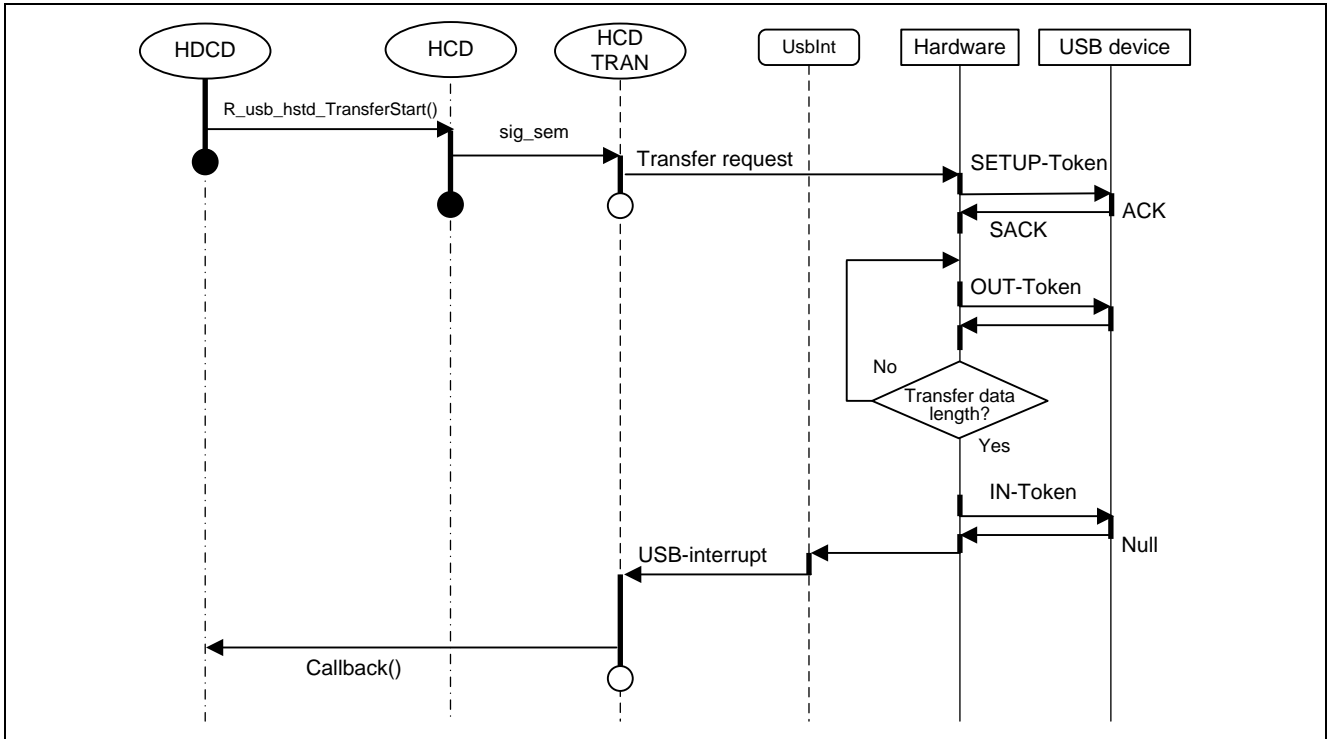
Figure 10.4   Control Write Sequence

10.3.3   Control Read Sequence
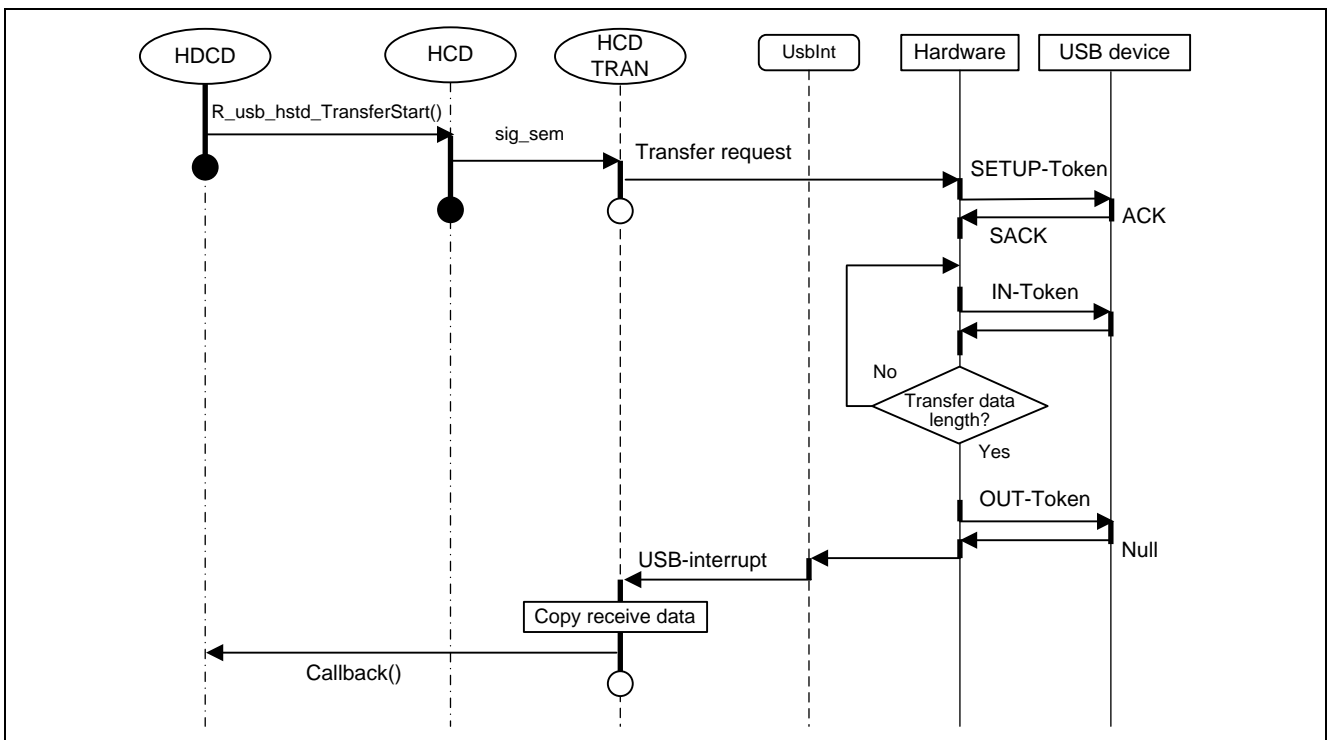
Figure 10.5   Control Read Sequence

11. Host Manager (MGR)

11.1   Basic Functionality

The MGR task supplements the functionality between the HCD module and HDCD. MGR provides the following functionality.

(1) Registering HDCD
(2) State management for connected device
(3) Enumeration of connected device
(4) Searching endpoint information in descriptors
(5) Starting HDCD
(6) Detach detection

11.2   Registering HDCD

APL uses a MGR API function to register HDCD.

11.3   Device State Management

When MGR receives a device state change request, it issues a device state change request to either HCD (if the device address in the request message is less than USB_DEVICEADDR) or to HUBCD (if the device address is equal to or greater than USB_DEVICEADDR).

11.4   USB Standard Requests

When MGR receives a device attach notification from HCD System, it issues a USB reset and then continues by performing enumeration. MGR saves the reset handshake result and performs a control transfer, as described below. At this time MGR assigns USB_DEVICEADDR to the device connected to port 0.

Note that MGR temporarily stores the descriptor information obtained from the device, and this information can be fetched by using the following MGR API functions.

(1) GET_DESCRIPTOR (DeviceDescriptor)
(2) SET_ADDRESS
(3) GET_DESCRIPTOR (ConfigurationDescriptor)
(4) SET_CONFIGURATION

11.5   Checking and Starting HDCD

MGR notifies HDCD via a callback of the information obtained during enumeration by using the GET_DESCRIPTOR request. HDCD checks the device information and determines if the connected device is operable within the callback function, and pass the result to MGR . If the response is operable, MGR performs enumeration (by executing a SET_CONFIGURATION request) and then starts HDCD.

11.6 MGR Outline Flowchart

The outline flowchart below shows MGR operation after receipt of a callback from HCD SYS.
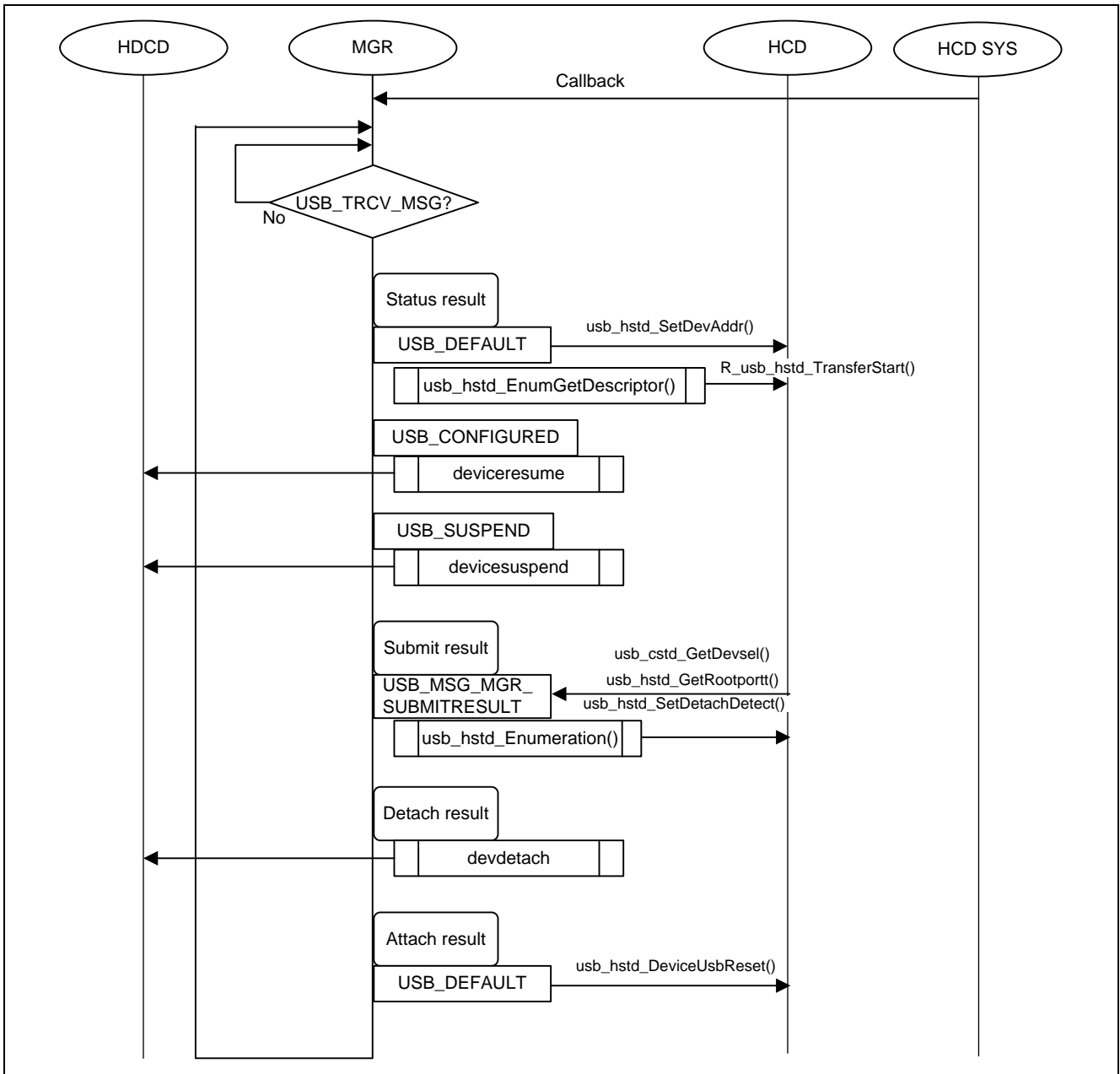
Figure 11.1 MGR Outline Flowchart 1

The outline flowchart below shows MGR operation after receipt of a request from APL/HDCD.
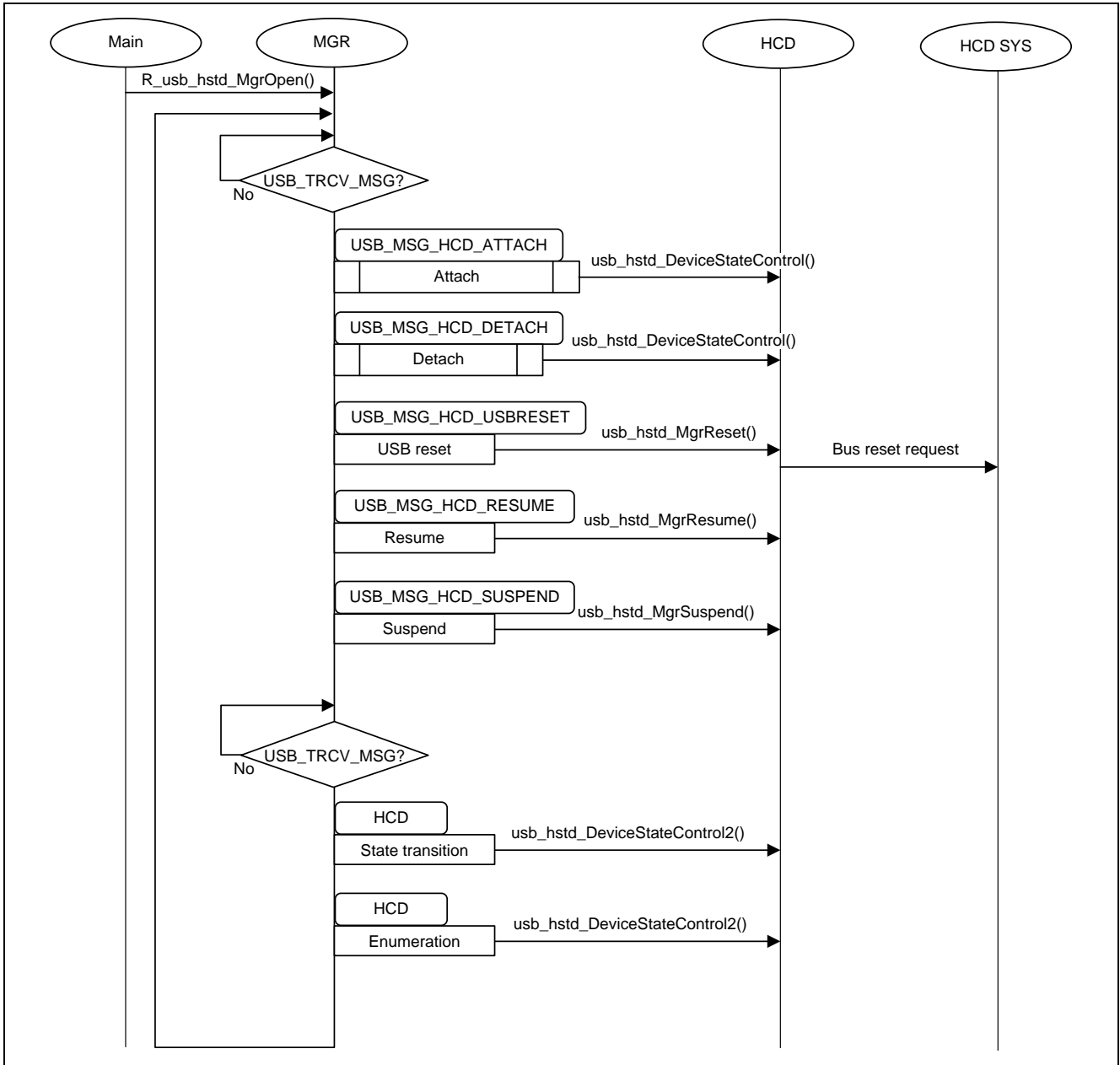
Figure 11.2 MGR Outline Flowchart 2

11.7   MGR API Functions

HDCD issues device state transition requests and checks device information by means of API functions. The API functions are contained in the file **r_usb_hDriverAPI.c**.

Table 11.1   List of MGR API Functions

	Function Name	Description
1	R_usb_hstd_MgrOpen()	Start MGR task
2	R_usb_hstd_MgrClose()	End MGR task
3	R_usb_hstd_DriverRegistration()	Register HDCD
4	R_usb_hstd_DriverRelease()	Release HDCD
5	R_usb_hstd_MgrChangeDeviceState()	Request change in state of connected device

11.7.1   Details of MGR API Functions

Table 11.2   R_usb_hstd_MgrOpen()

Name	Start MGR Task		
Call format	USB_ER_t R_usb_hstd_MgrOpen(void)		
Arguments	void	—	—
Return values	USB_ER_t	—	Error code
Description	This function starts the MGR task. It makes initial global variable settings; creates the task, mailboxes, and memory pools; and starts the task. The MGR task then waits for messages from HDCD or HCD.		
Notes	Calling this function starts the MGR task. Do not call it again after the task has started.		

Table 11.3   R_usb_hstd_MgrClose()

Name	End MGR Task		
Call format	USB_ER_t R_usb_hstd_MgrClose(void)		
Arguments	void	—	—
Return values	USB_ER_t	—	Error code
Description	This function ends the MGR task. It releases the mailboxes and memory pools, and terminates the task.		
Notes	Calling this function ends the MGR task. Do not call it again after the task has closed.		

Table 11.4   R_usb_hstd_DriverRegistration()

Name	Register HDCD		
Call format	void R_usb_hstd_DriverRegistration(USB_HCDREG_t *callback)		
Arguments	USB_HCDREG_t	*callback	
Return values	void	—	—
Description	This function registers HDCD. It registers the HDCD information, pipe information table, and callback functions for device state transitions.		
Notes	Update the number of registered drivers and register HDCD in a new area.		

Table 11.5   R_usb_hstd_DriverRelease()

Name	Release HDCD		
Call format	void R_usb_hstd_DriverRelease(uint8_t devclass)		
Arguments	uint8_t	devclass	Device address
Return values	void	—	—
Description	This function releases HDCD. The released driver area becomes an open area.		
Notes	The number of registered drivers does not change.		

Table 11.6   R_usb_hstd_MgrChangeDeviceState()

Name	Request Change in State of Connected Device		
Call format	USB_ER_t R_usb_hstd_MgrChangeDeviceState(USB_B_INFO_t complete, uint16_t msginfo, uint16_t devaddr)		
Arguments	USB_CB_INFO_t	complete	Callback function
	uint16_t	msginfo	Message information
	uint16_t	devaddr	Device address
Return values	USB_ER_t		Error code
Description	This function changes the state of a peripheral device connected to a USB port. It notifies MGR of the device state change request by transmitting a message, and MGR uses HCD or HUBCD to perform the necessary processing. When processing ends, the function notifies the high-level layer that the setting has been completed by using a callback function. To issue a request, specify one of the following arguments (msginfo) when calling the function. USB_GO_POWEREDSTATE: Request for transition from connected to disconnected state USB_DO_RESET_AND_ENUMERATION: Request for transition from connected to initialized for communication state (request to issue USB reset and perform enumeration) USB_PORT_ENABLE: Request for Vbus supply start USB_PORT_DISABLE: Request for Vbus shutdown USB_DO_GLOBAL_SUSPEND: Request for transition to suspended with remote wakeup state (suspending of all connected devices from port onward) USB_DO_SELECTIVE_SUSPEND: Request for transition to suspended with remote wakeup state (suspending of all connected devices from hub onward) USB_DO_GLOBAL_RESUME: Request for global resume (request for transition from suspended state to state preceding suspended state) (resume of all connected devices from hub onward) USB_DO_SELECTIVE_RESUME: Request for selective resume (request for transition from suspended state to state preceding suspended state) (resume of specified connected device from hub onward)		
Notes			

11.8   Details of Callback Functions

Table 11.7   classcheck

Name	Classcheck Callback Function		
Call format	(*USB_CB_CHECK_t)(uint16_t **);		
Arguments	uint16_t	**Table	Table[0] device descriptor Table[1] configuration descriptor Table[2] interface descriptor Table[3] descriptor check result Table[4] hub classification Table[5] port number Table[6] communication speed Table[7] device address
Return values	—	—	—
Description	Use this function to notify HDCD of the descriptor information and to send a response indicating whether HDCD is ready or not. Insert either of the following check results in Table[3] when sending a response. USB_DONE: HDCD ready USB_ERROR: HDCD not ready		
Notes			

Table 11.8   Other Callback Functions

Name	Other Callback Functions		
Call format	(*USB_CB_INFO_t)(uint16_t ,uint16_t);		
Arguments	uint16_t		NOARGUMENT: Not used
	uint16_t		NOARGUMENT: Not used
Return values	—	—	—
Description	classinit: Executed at MGR start. deviceconfig: Executed when Set_Configuration request issued. devicedetach: Executed at detach detection. devicesuspend: Executed at transition to suspend. deviceresume: Executed at transition to resume.		
Notes			

11.9   Structure Definitions

11.9.1   USB_MGRINFO_t Structure

The message structure transmitted to MGR by the usb_hstd_NotifAtorDetach(), usb_hstd_StatusResult(), usb_hstd_OvcrNotifiation(), R_usb_hstd_MgrChangeDeviceState()and usb_hstd_SubmitResult() functions is described below.

```
typedef struct {
  USB_MH_t     msghead;  // Message heater used by the OS
  uint16_t     msginfo;  // Message information used by USB-BASIC-F/W
  uint16_t     keyword;  // Sub-information (port number, pipe number, etc.)
  uint16_t     result    // Processing result
} USB_MGRINFO_t;
```

Table 11.9   Members of USB_MGRINFO_t Structure, when Used by usb_hstd_NotifAtorDetach(), usb_hstd_StatusResult() or usb_hstd_OvcrNotifiation()

Variable	Description
msghead	This message header is used by the OS, so the client should not make use of it.
msginfo	USB_MSG_MGR_AORDETACH (used by usb_hstd_NotifAtorDetach() function), USB_MSG_MGR_STATUSRESULT (used by usb_hstd_StatusResult() function), USB_MSG_MGR_OVERCURRENT (used by usb_hstd_OvcrNotifiation() function)
keyword	Port number
Result	Returns result of completed processing by HCD.

Table 11.10 Members of USB_MGRINFO_t Structure when Used by in usb_hstd_SubmitResult()

Variable	Description
msghead	This message header is used by the OS, so the client should not make use of it.
msginfo	USB_MSG_MGR_SUBMITRESULT(used by usb_hstd_SubmitResult() function)
keyword	Pipe number
Result	Returns result of completed processing by HCD.

Table 11.11 Members of USB_MGRINFO_t Structure when Used by R_usb_hstd_MgrChangeDeviceState()

Variable	Description
msghead	This message header is used by the OS, so the client should not make use of it.
msginfo	USB_GO_POWEREDSTATE, USB_DO_RESET_AND_ENUMERATION, USB_PORT_ENABLE, USB_PORT_DISABLE, USB_DO_GLOBAL_SUSPEND, USB_DO_SELECTIVE_SUSPEND, USB_DO_GLOBAL_RESUME, USB_DO_SELECTIVE_RESUME (used by R_usb_hstd_MgrChangeDeviceState() function)
keyword	Device address
Result	Returns result of completed processing by HCD.

11.10 Device State Management

11.10.1 State Transition Outline Sequence

An outline sequence of device state transitions is shown below.
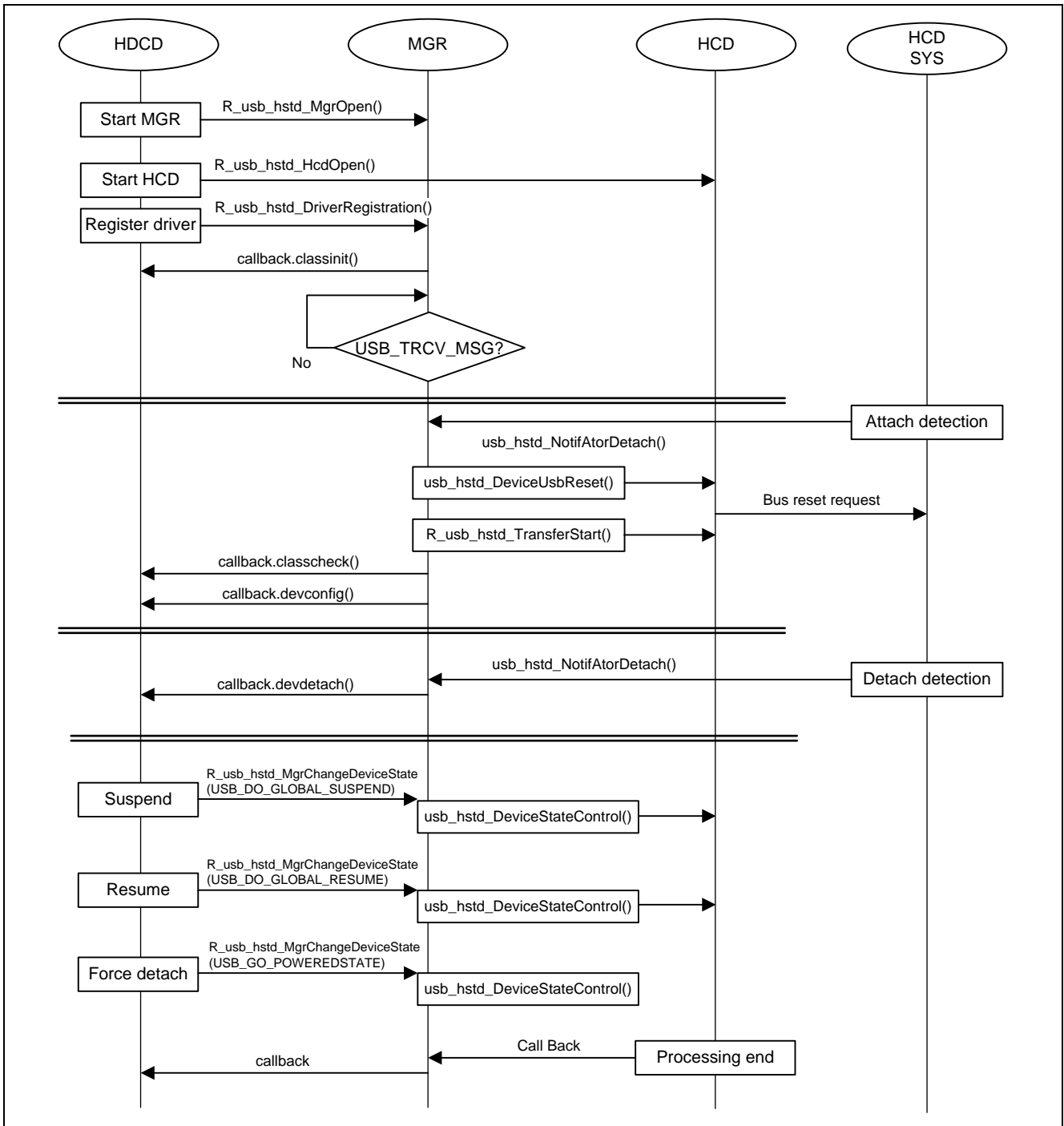
Figure 11.3 State Transition Sequence

11.10.2 Attach/Detach

The attach/detach sequence is shown below.
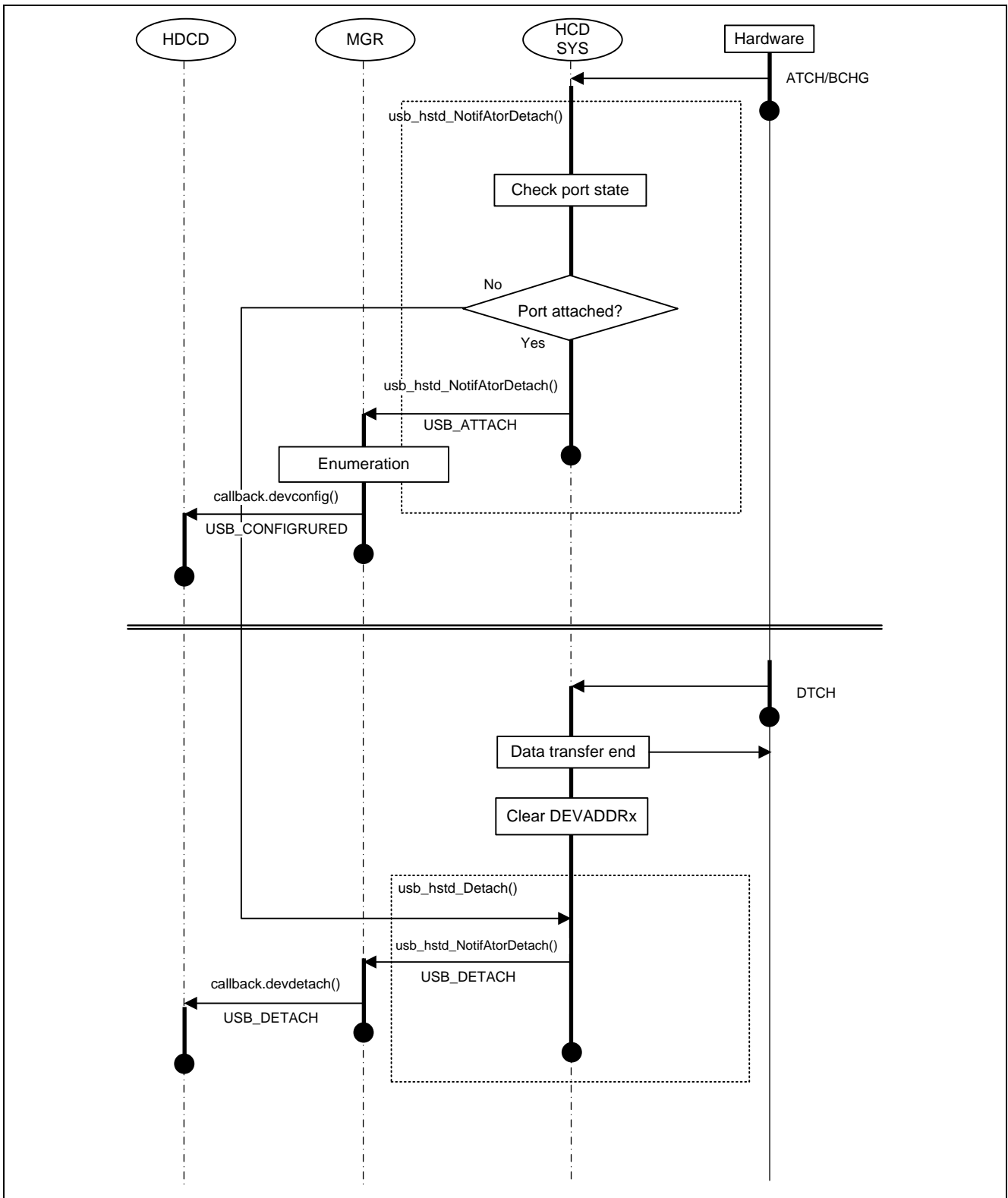
Figure 11.4 Attach/Detach

11.10.3 Suspend/Resume

The suspend/resume sequence is shown below.
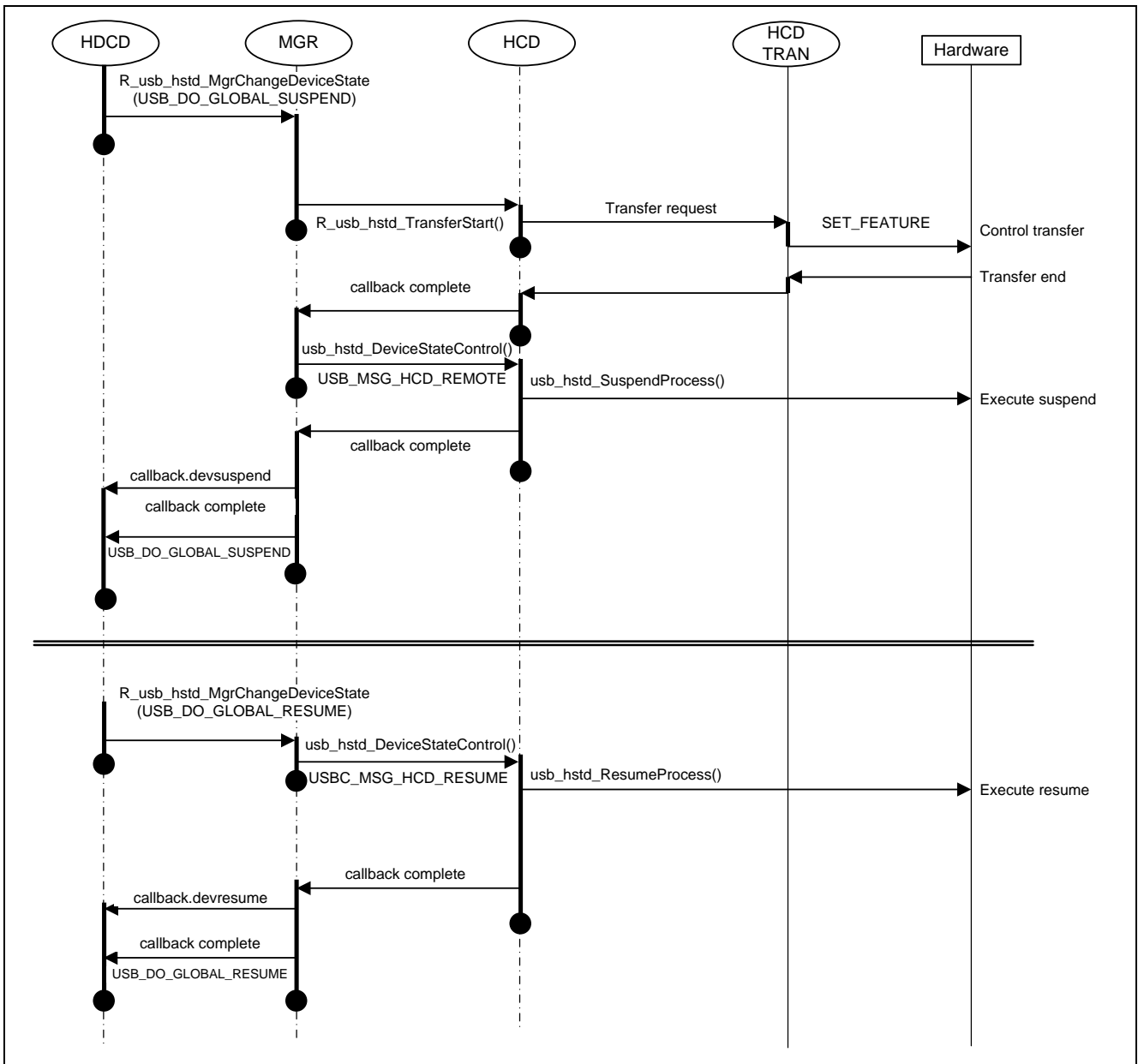
Figure 11.5 Suspend/Resume

11.11 Enumeration

The enumeration sequence is shown below.
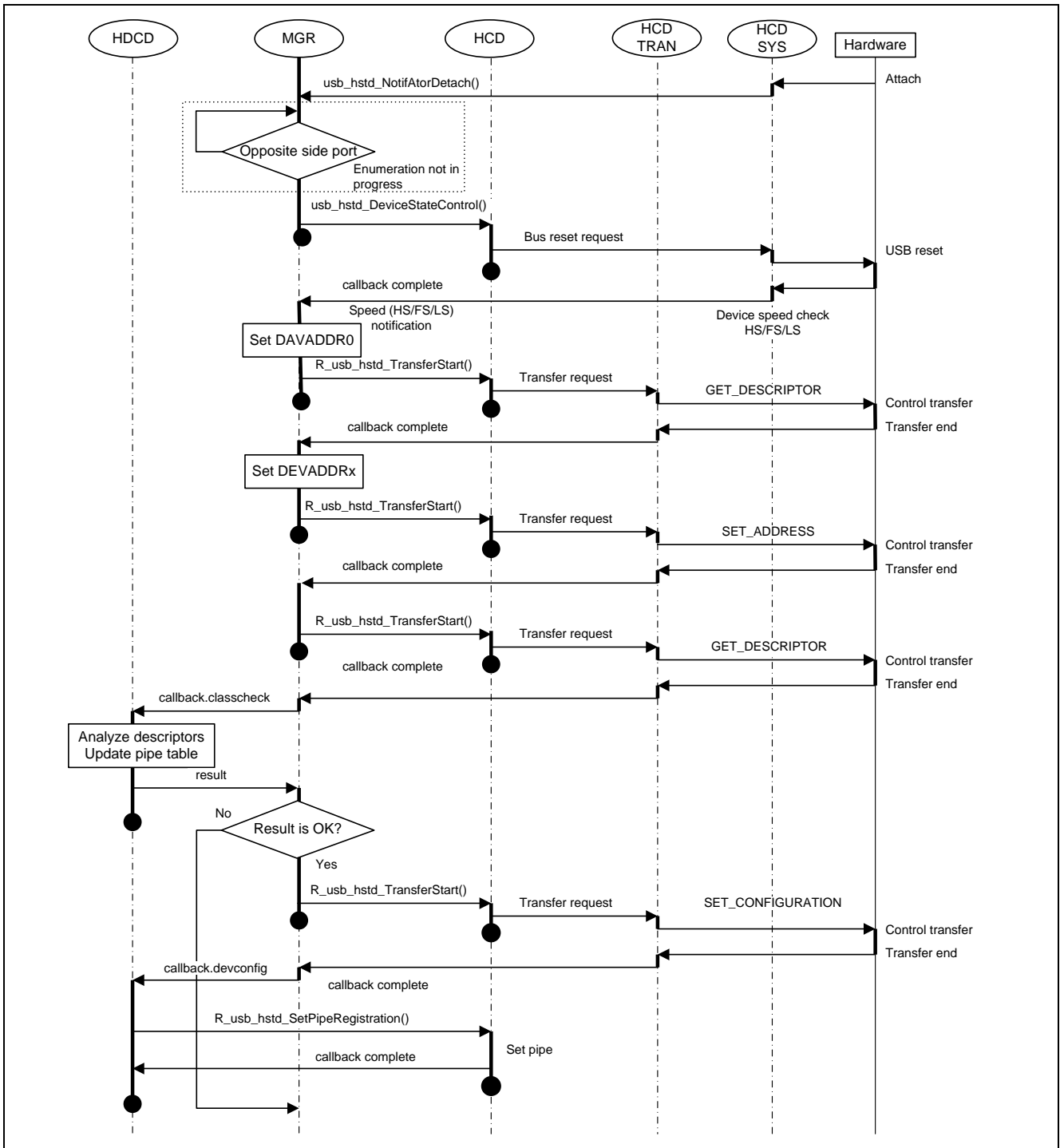
Figure 11.6 Enumeration

12. Hub Class Driver (HUBCD)

12.1   Basic Functionality

The HUBCD task manages the states of the down ports of the connected USB hub and supplements the functionality between HCD and HDCD. HUBCD provides the following functionality.

(1) State management of devices connected to the down ports of the USB hub
(2) Enumeration of devices connected to the down ports of the USB hub
(3) Starting HDCD

12.2   HUBCD API Functions

Table 12.1 lists the HUBCD API functions.

Table 12.1   List of HUBCD API Functions

	Function Name	Description
1	R_usb_hhub_Open()	Start HUBCD task
2	R_usb_hhub_Close()	Release HUBCD task (No call from the high-level layer is necessary because this task is usually executed by MGR.)
3	R_usb_hhub_Registration()	Register HUBCD
4	R_usb_hhub_ChangeDeviceState()	Request change in state of device connected to hub (µITRON version only)
5	R_usb_hhub_GetHubInformation()	Get hub information
6	R_usb_hhub_GetPortInformation()	Get hub port information

12.2.1   Details of HUBCD API Functions

Table 12.2   R_usb_hhub_Open()

Name	Start HUBCD Task		
Call format	USB_ER_t R_usb_hhub_Open(uint16 devaddr, uint16 data2)		
Arguments	uint16_t	devaddr	USB hub device address
	uint16_t	data2	Not used
Return values	USB_ER_t	—	Error code
Description	This function starts the HUBCD task. It makes initial global variable settings; creates the task, mailboxes, and memory pools; and starts the task. The HUBCD task waits for a request from MGR. No call from the high-level layer is necessary because this task is usually executed by MGR.		
Notes	Do not call this function after the HUBCD task has started.		

Table 12.3   R_usb_hhub_Close()

Name	End HUBCD Task		
Call format	USB_ER_t R_usb_hhub_Close(uint16 hubaddr, uint16 data2)		
Arguments	uint16_t	hubaddr	USB hub device address
	uint16_t	data2	Not used
Return values	USB_ER_t	—	Error code
Description	This function ends the HUBCD task. It releases the mailboxes and memory pools, and terminates the task. No call from the high-level layer is necessary because this task is usually executed by MGR.		
Notes	Do not call this function after the HUBCD task has started.		

Table 12.4   R_usb_hhub_Registration()

Name	Register HUBCD		
Call format	void R_usb_hhub_Registration(void)		
Arguments	void		
Return values	void	—	—
Description	This function registers HUBCD information in a table managed by HCD. This function updates the number of registered drivers managed by HCD and registers HUBCD in a new area.		
Notes			

Table 12.5   R_usb_hhub_ChangeDeviceState()

Name	Request Change in State of Device Connected to Hub		
Call format	USB_ER_t R_usb_hhub_ChangeDeviceState(USB_CB_INFO_t complete, uint16_t msginfo, uint16_t devaddr)		
Arguments	USB_CB_INFO_t	complete	Callback function
	uint16_t	msginfo	Message information
	uint16_t	devaddr	Device address
Return values	USB_ER_t	—	Error code
Description	This function requests a change to the state of a device connected to the hub. #define USB_MSG_HCD_ATTACH: Request for transition to connected state #define USB_MSG_HCD_DETACH: Request for transition from connected to disconnected state #define USB_MSG_HCD_USBRESET: Request to issue USB reset #define USB_MSG_HCD_SUSPEND: Request for transition to suspended state #define USB_MSG_HCD_RESUME: Request to issue resume signal #define USB_MSG_HCD_REMOTE: Request for transition to suspended with remote wakeup state #define USB_MSG_HCD_VBON: Vbus output start request #define USB_MSG_HCD_VBOFF: Vbus output end request #define USB_MSG_HCD_CLR_STALL: Stall clear request		
Notes	Used by µITRON version only.		

Table 12.6   R_usb_hhub_GetHubInformation()

Name	Get Hub Information		
Call format	uint16_t R_usb_hhub_GetHubInformation(uint16_t hubaddr, USB_CB_t complete)		
Arguments	uint16_t	hubaddr	USB hub device address
	USB_CB_t	complete	Callback function
Return values	uint16_t	—	Error code
Description	This function obtains the hub descriptor information.		
Notes			

Table 12.7   R_usb_hhub_GetPortInformation()

Name	Get Hub Port Information		
Call format	uint16_t R_usb_hhub_GetPortInformation(uint16_t hubaddr, uint16_t port, USB_CB_t complete)		
Arguments	uint16_t	hubaddr	USB hub device address
	uint16_t	port	Port number
	USB_CB_t	complete	Callback function
Return values	uint16_t	—	Error code
Description	This function obtains the hub port state.		
Notes			

12.3   Down Port State Management

When the HUBCD task is launched, it performs the following actions on all down ports to determine the device connection state of each down port.

1. Enable port power (HubPortSetFeature: USB_HUB_PORT_POWER)
2. Initialize port (HubPortClrFeature: USB_HUB_C_PORT_CONNECTION)
3. Get port status (HubPortStatus: USB_HUB_PORT_CONNECTION)

12.4   Connecting Devices to Down Ports

When HUBCD receives a device attach notification from the USB hub, it issues a USB reset signal to the USB hub (HubPortSetFeature: USB_HUB_PORT_RESET). Next, it uses the resources of the MGR task to enumerate the connected device. HUBCD stores the result of the reset handshake and assigns an address, starting with USB_DEVICEADDR + 1 for the first device connected and becoming sequentially higher with each successive device.

12.5 Class Requests

HUBCD supports the class requests listed below.

Table 12.8 USB Hub Class Requests

Request	Implementation	Function	Functionality
ClearHubFeature	No		
ClearPortFeature	Yes	usb_hhub_ PortClrFeature	USB_HUB_PORT_ENABLE USB_HUB_PORT_SUSPEND USB_HUB_C_PORT_CONNECTION USB_HUB_C_PORT_ENABLE USB_HUB_C_PORT_SUSPEND USB_HUB_C_PORT_OVER_CURRENT USB_HUB_C_PORT_RESET
ClearTTBuffer	No		
GetHubDescriptor	Yes	R_usb_hhub_ GetHubInformation	Get descriptors
GetHubStatus	No		
GetPortStatus	Yes	R_usb_hhub_ GetPortInformation	Get port status
ResetTT	No		
SetHubDescriptor	No		
SetHubFeature	No		
SetPortFeature	Yes	usb_hhub_ PortSetFeature	USB_HUB_PORT_POWER USB_HUB_PORT_RESET USB_HUB_PORT_SUSPEND USB_HUB_C_PORT_ENABLE
GetTTState	No		
StopTT	No		

12.6 Down Port Device State Management

12.6.1 Hub Attach/Detach

The attach/detach sequence is shown below.
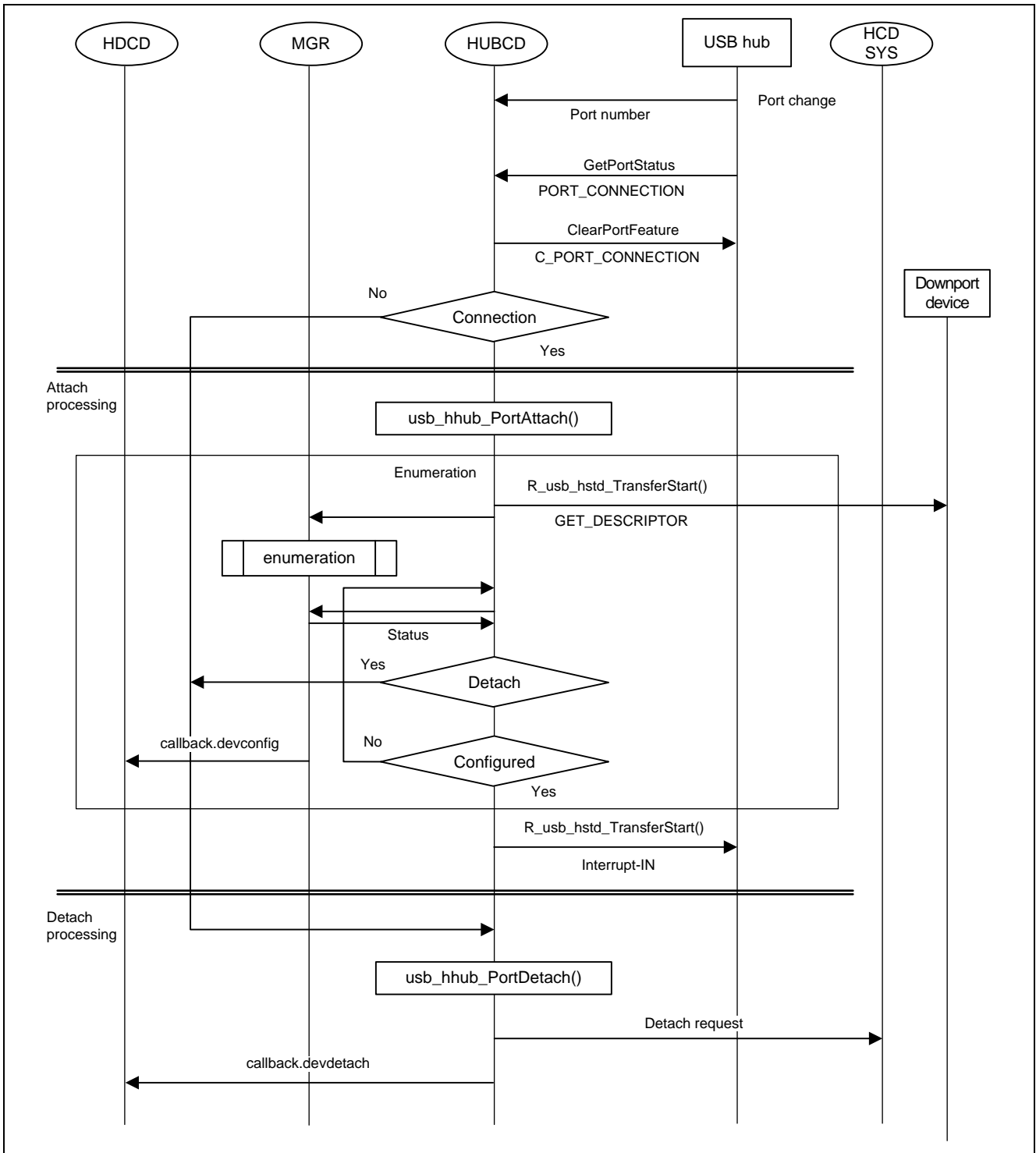
Figure 12.1 Hub Attach/Detach

12.7   Down Port State Management and Enumeration

The enumeration sequence is shown below.
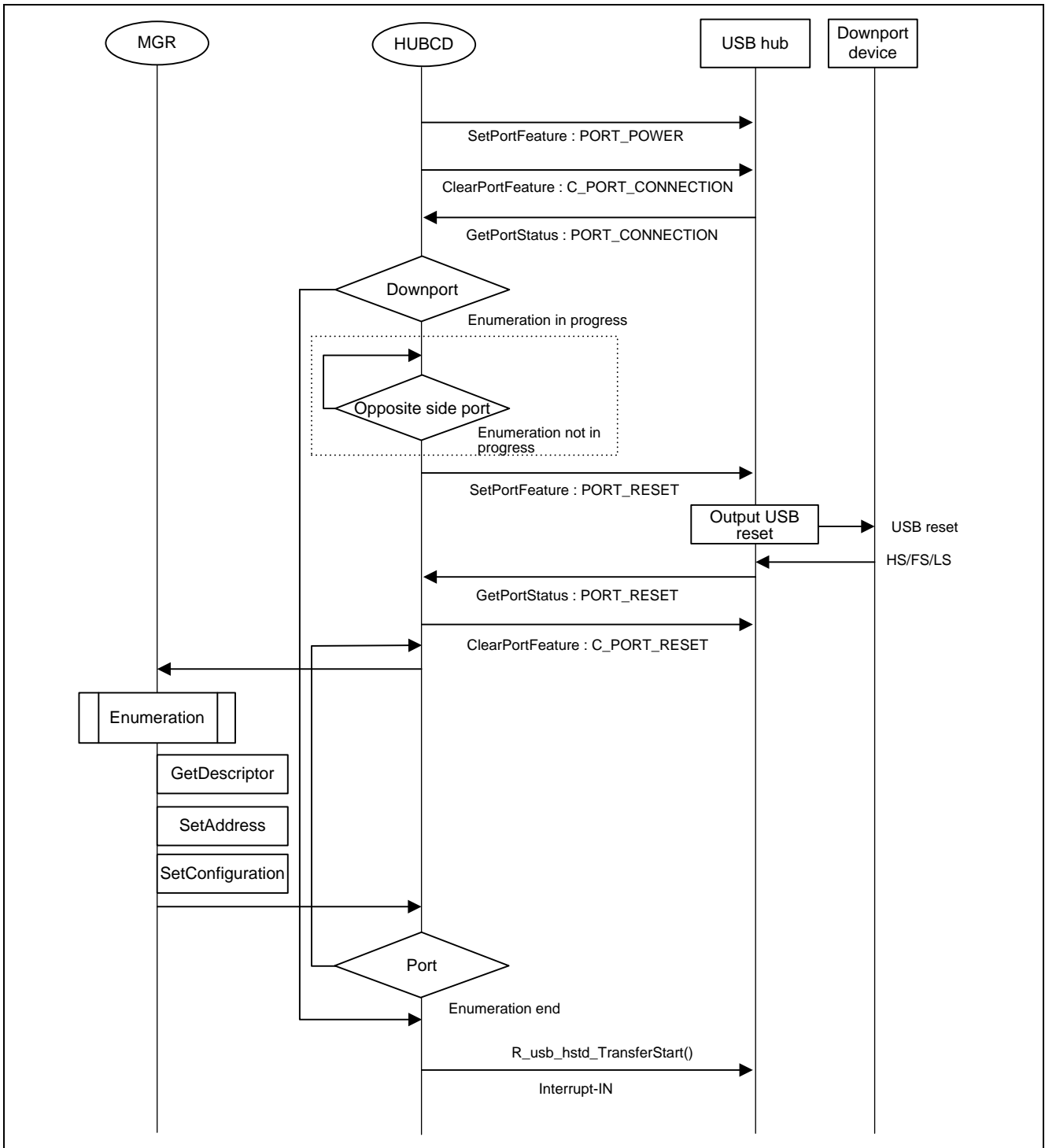
Figure 12.2   Down Port Device Enumeration

13. Data Transfer

13.1　Overview

USB-BASIC-F/W can perform data transfers by notifying HCD of the data transmit/receive user buffer. Data transfers use the customer's own functional specifications, and settings such as transfer method, communication start/stop timing, and buffer configuration must be modified as necessary to match the system under development.

13.1.1　Basic Specifications

USB-BASIC-F/W performs data transfers between the user buffer specified in a structure and the hardware according to the information specified in the pipe information table.

Note:　The version of USB-BASIC-F/W with EHCI support uses a pipe information table to maintain interface compatibility with the version of USB-BASIC-F/W for R8A6659x, but it does not make use of the settings specific to the R8A6659x hardware that are contained in the pipe information table (for example, FIFO buffer size).

13.1.2　Feedback of Transfer Result

When a data transfer completes, USB-BASIC-F/W notifies HDCD of data transfer end by using the registered callback function.

The notification sent by USB-BASIC-F/W to HDCD can be any one of the following nine communication results.

- USB_CTRL_END: Successful control transfer end
- USB_DATA_NONE: Successful data transfer end
- USB_DATA_OK: Successful data receive end
- USB_DATA_SHT: Successful data receive end, but completed with less than the specified data length
- USB_DATA_OVR: Receive data size over specified length
- USB_DATA_ERR: No response or overrun/underrun error detected
- USB_DATA_STALL: STALL response or MaxPacketSize error detected
- USB_DATA_STOP: Forced end of data transfer
- USB_DATA_TMO*: No callback when forced end occurs due to timeout.

13.1.3　Note on Data Transfer

Run a submit function to ensure that the interval duration is maintained during isochronous data transfer.

13.1.4　Note on Data Reception

When a short packet is received, the data length of the remaining portion still to be received is stored in **tranlen** and the transfer ends.

13.2   USB Communication Structure (USB_UTR_t Structure)

The structure that is passed as an argument of the R_usb_hstd_TransferStart() function is described below. USB communication with a peripheral device is enabled by notifying HCD with this structure.

```
struct USB_SUTR   {
  USB_MH_t    msghead;        //  Message heater used by the OS
  uint16_t    msginfo;        //  Message information used by USB-BASIC-F/W
  uint16_t    keyword;        //  Sub-information (port number, pipe number, etc.)
  void        *tranadr;       //  Transfer data start address
  uint32_t    tranlen;        //  Transfer data length
  uint16_t    *setup;         //  Setup packet (only host control transfer)
  uint16_t    status;         //  Transfer end status
  uint16_t    pipectr;        //  Pipe control register state
  USB_CB_t    complete;       //  Callback function at processing end
  uint8_t     errcnt;         //  Error count
  uint8_t     segment;        //  Segment code
}USB_UTR_t;
```

Table 13.1   Members of USB_UTR_t Structure

Variable	R/W	Description
msghead	—	This message header is used by the OS. Do not make use of it.
msginfo	R	Message classification Specifies the content of the request. This is set by USB-BASIC-F/W by means of an API function. It is set to USB_MSG_HCD_SUBMITUTR when performing USB communication.
keyword	R	Subcode Specifies the pipe number when performing USB communication.
*tranadr	R	USB communication buffer address Provides notification of the USB communication buffer address.
tranlen	R/W	USB communication data length Provides notification of the USB communication data length. The transfer size specified should be less than the user buffer size.
*setup	R	Setup packet data Provides notification to HCD of the setup packet request and device address for control transfer.
status	W	USB communication status HCD responds with the USB communication result.
pipectr	—	Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
complete	R	Callback function Specifies the address of the function to be executed when USB communication ends. Use the following type declaration for the callback function. typedef void (*USB_CB_t)(USB_UTR_t*);
errcnt	—	Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
segment	—	Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.

(1)   USB Communication Buffer Address

- Reception or ControlRead transfer: Specifies the buffer address for storing receive data.
- Transmission or ControlWrite transfer: Specifies the buffer address for storing transmit data.
- NoDataControl transfer: Ignored even of specified.

Note: The USB basic firmware uses the buffer at the specified address as direct transfer memory, so the area must be secured and must not be read or written until the transfer completes

(2)   USB Communication Data Length

- Reception or ControlRead transfer: Stores the receive data length.
- Transmission or ControlWrite transfer: Stores the transmit data length.
- NoDataControl transfer: Set to 0.

The remaining transmit/receive data length is stored after USB communication ends. In the case of a host function control transfer, the remaining data length for the Data stage is stored.

Note: For isochronous transfers, always set the USB communication data length to a value smaller than the maximum packet size. Operation cannot be guaranteed if a setting value larger than the maximum packet size is used.

(3)   Setup Packet Data

In a control transfer by the R_usb_hstd_TransferStart() function, the structure member (*setup) is a USB request data table, as shown below.

Specify the uint16_t[5] array table address for *setup.

Table 13.2   setup_packet Array

Name	Value	
USBREQ	bRequest	bmRequestType
USBVAL	wValue	
USBINDX	wIndex	
USBLENG	wLength	
USBADDR	Device Address	

(4)   USB Communication Status

The following status information is returned.

- USB_CTRL_END: Successful control transfer end
- USB_DATA_NONE: Successful data transfer end
- USB_DATA_OK: Successful data receive end
- USB_DATA_SHT: Successful data receive end, but completed with less than the specified data length
- USB_DATA_OVR: Receive data size over specified length
- USB_DATA_ERR: No response or overrun/underrun error detected
- USB_DATA_STALL: STALL response or MaxPacketSize error detected
- USB_DATA_STOP: Forced end of data transfer
- USB_DATA_TMO*: No callback when forced end occurs due to timeout.

(5)   PIPECTR Register

Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.

(6)   Segment Information

Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.

13.3 Pipe Definitions

13.3.1 Overview

HDCD must maintain a pipe information table containing appropriate pipe settings for the relevant class driver. HDCD must also register a pipe information table when registering a driver.

Note: The version of USB-BASIC-F/W with EHCI support uses a pipe information table to maintain interface compatibility with the version of USB-BASIC-F/W for R8A6659x, but it does not make use of the settings specific to the R8A6659x hardware that are contained in the pipe information table (for example, FIFO buffer size). The unused settings are ignored.

13.3.2 Pipe Information Table

A pipe information table comprises the following six items (uint16_t × 6).

1. Pipe window select register
2. Pipe configuration register (Contains settings ignored by USB-BASIC-F/W with EHCI support.)
3. Pipe buffer specification register (Setting value ignored by USB-BASIC-F/W with EHCI support.)
4. Pipe max packet size register
5. Pipe cycle control register (Setting value ignored by USB-BASIC-F/W with EHCI support.)
6. FIFO port use method (Setting value ignored by USB-BASIC-F/W with EHCI support.)

13.3.3 Pipe Definitions

The sample pipe definitions included with USB-BASIC-F/W have the configuration shown below. The definition items that can be included in the information table are macro defined by r_usb_cDefUSBIP.h.

Example:

```
uint16_t usb_gpstd_SmplEpTbl1 [] = {          ← Registered information table
  USB_PIPE1,                                  ← Pipe definition item 1
  USB_BULK|USB_BFREOFF|USB_DBLBOFF|USB_CNTMDOFF|USB_SHTNAKON|USB_DIR_P_OUT|USB_EP1,
                                              ← Pipe definition item 2
  (uint16_t)USB_BUF_SIZE(512u)|USB_BUF_NUMB(8u),  ← Pipe definition item 3
  USB_SOFT_CHANGE,                            ← Pipe definition item 4
  USB_IFISOFF|USB_IITV_TIME(0u),              ← Pipe definition item 5
  USB_CUSE,                                   ← Pipe definition item 6
          :
          :
  USB_PDTBLEND
};
```

(1)　Pipe Definition Item 1

Specifies the value to be set in the pipe window select register.

- Pipe select: Specify the pipe selection (PIPE1 to PIPE30).

Example: Pipe 1
     USB_PIPE1

(2) Pipe Definition Item 2

Specifies the pipe configuration settings.

- Transfer type: Specify one of USB_BULK, USB_INT, or USB_ISO.
- BRDY operation specification: Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
- Double buffer mode: Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
- Continuous transmit/receive mode: Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
- SHTNAK operation specification: Not used by USB-BASIC-F/W with EHCI support, so the setting value is ignored.
- Transfer direction: Specify either USB_DIR_H(P)_OUT or USB_DIR_H(P)_IN.
- Endpoint number: Specify the endpoint number (EP1 to EP15).

Example: Bulk transfer, BFRE off, double buffer, continuous transmit/receive, OUT direction, EP2
        Host mode:
        USB_BULK | USB_BFREOFF | USB_DBLBON | USB_CNTMDOFF | USB_SHTNAKOFF |
        USB_DIR_H_OUT | USB_EP2

(3) Pipe Definition Item 3

Setting value ignored by USB-BASIC-F/W with EHCI support.

(4) Pipe Definition Item 4

Specifies the pipe maximum packet size register and device address settings.

- Device address: Specify the device address. In the sample application the initial value is set to USB_NONE because the value is later changed by software.
- *Maximum packet size: Specify the maximum packet size of the pipe. In the sample application the initial value is set to USB_NONE because the value is later changed by software.

Example 1: Maximum packet size 64, device address 3
        DEV_ADDR(3) | MAX_PACKET(64)

(5) Pipe Definition Item 5

Setting value ignored by USB-BASIC-F/W with EHCI support.

(6) Pipe Definition Item 6

Setting value ignored by USB-BASIC-F/W with EHCI support.

13.3.4   Limitations Affecting Pipe Definition Settings

- Use device class to specify transfer unit communication synchronization.
- Different pipes may not be set to the same pipe number at the same time (pipe definition item 1). (Use the R_usb_hstd_SetPipeRegistration() function to change the pipe setting as necessary.)

Note: Do not fail to write USB_PDTBLEND at the end of the table.

13.3.5   Basic Functionality

HDCD uses the R_usb_hstd_DriverRegistration() function to register the pipe information table.

The pipe information table must be updated to match the connected device.

Analyze the descriptor (EndpointDescriptor) obtained from the device and update the pipe information table accordingly.

(1)   Pipe Information Update Functions

The sample code includes the usb_hstd_ChkPipeInfo() function, which updates the data of the pipe information table based on the EndpointDescriptor, and the usb_hstd_SetPipeInfo() function, which makes settings in the pipe information table according to the updated information.

- Transfer type: One among USB_BULK, USB_INT, and USB_ISO is specified.
- Transfer direction: Either USB_DIR_H_OUT or USB_DIR_H_IN is specified.
- Endpoint number: The endpoint number (EP1 to EP15) is specified.
- Maximum packet size: The maximum packet size for the pipe is specified.
- Interval duration: The interval value is specified.

13.4 Data Transmit Operation

USB-BASIC-F/W uses the following procedure to transmit data.

[1] HDCD uses an API function to issue a data transfer request.

[2] Determine the pipe number and start data transfer.

[3] Set the data to be transmitted in the hardware and start the transfer.

[4] When the transfer completes, notify HCD TRAN by generating an interrupt in the interrupt handler.

[5] After a transfer end interrupt is received, call the callback function.
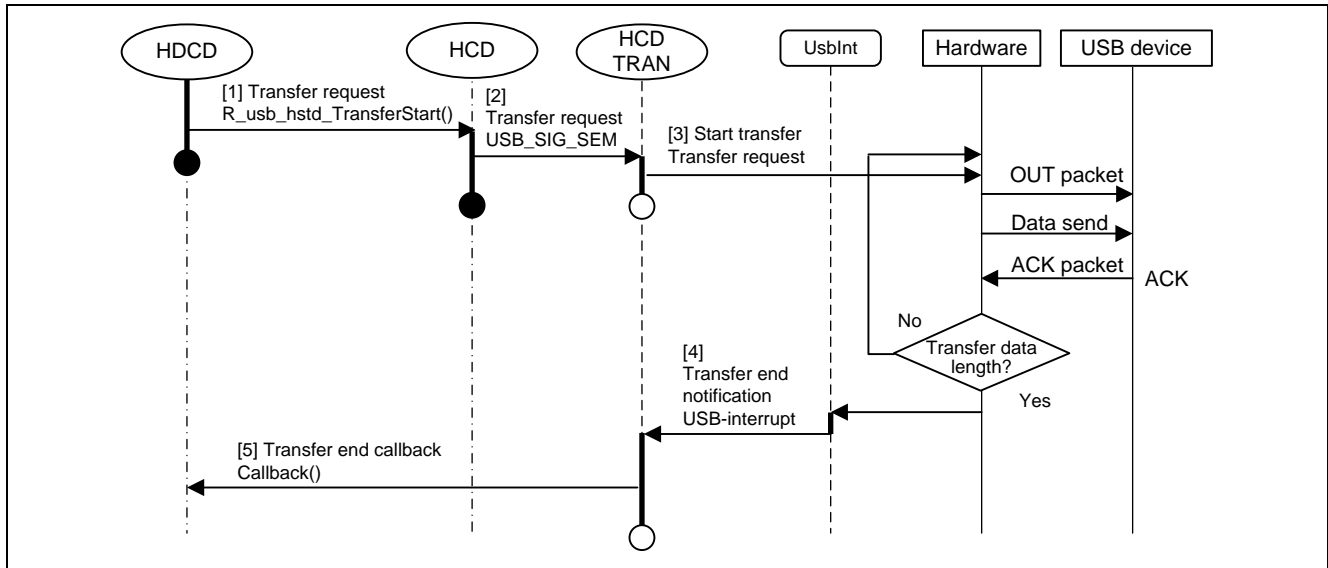
Figure 13.1 Data Transmit Outline Flowchart

13.5   Data Receive Operation

USB-BASIC-F/W uses the following procedure to perform receive data.

[1] and [2] Same as data transmit.

[3] Set receive buffer in hardware and start reception.

[4] When the transfer completes, notify HCD TRAN by generating an interrupt in the interrupt handler.

[5] After a transfer end interrupt is received, copy the receive data to the user buffer.

[6] Call the callback function.
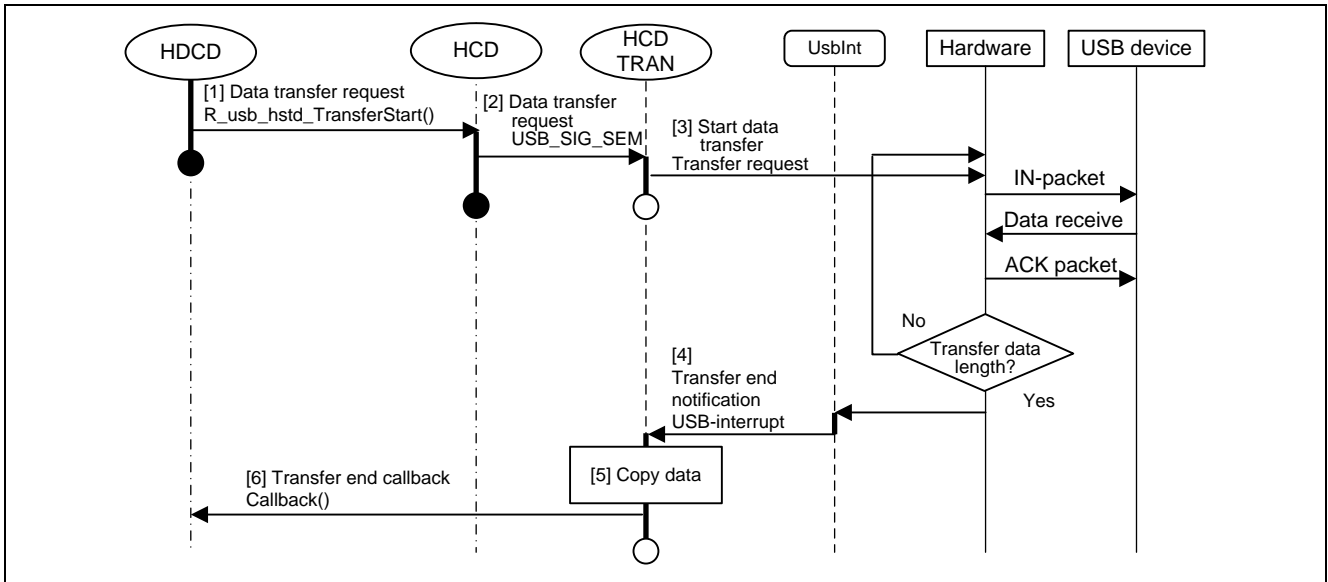
Figure 13.2   Data Receive Outline Flowchart

14. Restrictions

USB-BASIC-F/W for the USB 2.0 host controller IP with EHCI support is subject to the following restrictions.

1. Isochronous transfer may drop packets in a system where the MCU is subject to a high load. (For example, it may not be possible to set in the prescribed time packets transmitted by isochronous transfer.)
2. The structures contain members of different types. (Depending on the compiler, the address alignment of the structure members may be shifted.)
3. HDCD must be prepared by the customer.
4. USB-BASIC-F/W does not support suspend/resume of the connected hub and devices connected to the hub's down ports.
5. The maximum number of hub down port devices is four. If support for more than four down ports is required, the customer must make modifications to HUBCD.
6. The version of USB-BASIC-F/W with EHCI support uses a pipe information table to maintain interface compatibility with the version of USB-BASIC-F/W for R8A6659x, but it does not make use of the settings specific to the R8A6659x hardware that are contained in the pipe information table (for example, FIFO buffer size).
7. Due to restriction item 6, the maximum number of connected devices and the maximum number of pipes are restricted as follows, due to limitations imposed by the pipe information table specifications.
    — Maximum number of connected devices: 14
    — Maximum number of pipes: 30
8. When using isochronous transfer, the transfer data for a single transfer request (R_usb_hstd_TransferStart()) may not exceed the maximum packet size.
9. High-speed isochronous transfer does not support the high-bandwidth specification.
10. USB-BASIC-F/W does not include a function to synchronize with the audio clock of an audio device in cases where an audio class driver with an isochronous out transfer function is used as the HDCD. Therefore, issues such as noise may arise during isochronous out transfer with an audio device. Functionality for synchronization with the audio clock of an audio device must be implemented as part of the customer's system.
11. USB-BASIC-F/W with EHCI support does not include a function for adjusting the SOF interval.
12. USB-BASIC-F/W with EHCI support uses the USB communication buffer memory specified by the R_usb_hstd_TransferStart() function as direct transfer memory for the hardware. After the R_usb_hstd_TransferStart function is called, the high-level software must not access the USB communication buffer memory until the transfer completes. In addition, during receive transfers do not perform write access even outside the USB communication buffer if write access near either end of the buffer area involves loading data from both ends of the USB communication buffer into cache memory. Doing so could disrupt the consistency of cache memory and physical memory. To avoid this problem, use the following workarounds.
    — Do not perform write accesses to the memory near the USB communication buffer during transfers.
    — Align both ends of the USB transfer buffer with cache memory boundaries.
    — Use a non-cached area as the USB communication buffer.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	**Summary**
1.01	Sep.27.11	—	
1.02	Sep.7.12	P2	1.2 Related Documents Document name amended. 　　USB2.0 Host Controller IP with Renesas EHCI 　　HardwareManual 　-> Add:SH778 Group User's Manual(Hardware part)
		P3	2.1 Features of USB-BASIC-F/W Description amended 　Delete API for host function data communication
		P3	2.4 Task Structure Description amended 　　The USB 2.0 host controller IP with Renesas EHCI 　-> USB Basic Firmware
		P7	3.2 Making Changes to USB-BASIC-F/W API Function name amended 　usb_cpustd_DelayXms() -> usb_cstd_DelayXms() 　usb_cpustd_Delay1us() -> usb_cstd_Delay1us() 　usb_cpu_UsbintInit() -> usb_cstd_UsbintInit()
		P11	5.1 Overview Description amended 　r_usbc_DefUsrPb.h -> r_usbc_cDefUsrPb.h
		P11	5.2.1 USB Port Specification Description amended 　USBC_1PORT_PP -> USB_1PORT_PP 　USBC_2PORT_PP -> USB_2PORT_PP
		P12	5.3.2 Device Address, Description amended 　Use default value for example. device address 2 -> 1
		P12	5.3.3 Number of Hub Down Ports (when Host Function Selected) Description amended 　USBC_HUBDOWNPORT -> USB_HUBDOWNPORT
		P14	5.4.8 Specification of EHCI Timeout Duration Description amended 　Use default value for example. timeout 3000 -> 3
		P14	5.5 OHCI User-Defined Information File (r_usb_hOhciDefUsr.h) 　Seventh item description amended 　　maximum isochronous data transfer size 　-> maximum isochronous frame count
		P15	5.5.7 Specification of OHCI Maximum Isochronous frame count 　Title and Description amended
		P19	Added 6.4 Sections
		P21	Table 6.3 System Calls Description amended 　USB_DEL_TSK Releases a task -> Deletes a task 　USB_DEL_MBX Releases a mailbox -> Deletes a mailbox 　USB_DEL_MPL Releases a memory pool. -> Deletes a memory pool. 　USB_DEL_SEM Releases a semaphore -> Deletes a semaphore

		P26	Table 7.5　R_usb_hstd_SetPipeRegistration() Description amended 　Since the USB 2.0 host controleer IP with EHCI support -> 　Since the Basic Firmware with EHCI support
		P30	7.8.2 Information Registered in HCD (USB_HCDREG_t Structure) Description added 　To register is device as a device class **that supports all devices**,
		P33	10.1.1 Basic Specifications Description amended 　using the setup **buffer** and user buffer specified in the structure as transfer memory. 　-> using the setup **information** and user buffer specified in the structure as transfer memory.
		P38	11.5 Checking and Starting HDCD Description amended
		P44	11.9.1　USB_MGRINFO_t Structure Description amended 　The descriptions "hstd_DeviceDetach()", "usb_hstd_DeviceUsbReset()", "usb_hstd_DeviceResume()", and "usb_hstd_SetReDetachControl()", usb_hstd_NotifAtorDetach(), are deleted. 　The descriptions "R_usb_hstd_MgrChangeDeviceState()" and "usb_hstd_SubmitResult() functions" are added.
		P44	Table 11.9, Table 11.10 and Table 11.11 Table name amended
		P55	13.1.1 Basic Specifications The description "The unused settings are ignored." Is deleted.
		P55	13.1.4 Note on Data Reception The description "When the received data is larger than…." Is deleted.
		P56	Table 13.1 Members of USB_UTR_t Structure Description amended 　tranlen W -> tranlen　　R/W
		P57	13.2 USB Communication Structure (USB_UTR_t Structure) Description Added in "(1)USB Communication Buffer Address" 　Note:　　　The USB basic firmware uses the buffer at the specified address as direct transfer memory, so the area must be secured until the transfer completes
		P64	14. Restrictions Description amended in item 4 　Once the hardware starts the transfer 　-> After the R_usb_hstd_TransferStart function is called
1.03	Sep.10.12	P16	6.2.1 Folder Structure "smpl" folder is renamed to "SMPL". "Project" folder.is deleted
		P18	Table 6.1 List of Files "r_usb_cKernelId.h" is added The path of "SMPL" and "hubd" folders are amended
		P58	13.2 USB Communication Structure (USB_UTR_t Structure) Description Amended in "(1)USB Communication Buffer Address"

P20	6.4 Sections	
	The restriction of assignment of isochronous transfer descriptor is added.	
P1-P66	This application note number is changed.	

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.

RENESAS

Renesas Electronics Corporation

http://www.renesas.com