R01AN3428EJ0110
Rev.1.10
Sep 30, 2016

RZ/A1H Group

USB Host Communications Device Class Driver (HCDC)

Introduction

This application note describes USB Host Communication Device Class Driver (HCDC). This module operates in combination with the USB basic firmware (USB-BASIC-F/W). It is referred to below as the USB HCDC.

Target Device

RZ/A1H Group

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

Related Documents

1. Universal Serial Bus Revision 2.0 specification
   http://www.usb.org/developers/docs/
2. USB Class Definitions for Communications Devices Revision 1.2
3. USB Communications Class Subclass Specification for PSTN Devices Revision 1.2
   http://www.usb.org/developers/docs/
4. RZ/A1H Group,RZ/A1M Group User's Manual: Hardware (Document No.R01UH0403EJ)
5. RZ/A1H Group USB Host and Peripheral Interface Driver (Document No.R01AN3291EJ)
6. RZ/A1H Group Downloading Program to NOR Flash Memory Using ARM® Development Studio 5 (DS-5™) Semihosting Function (for GENMAI) (Document No.R01AN1957EJ)
7. RZ/A1H Group I/O definition header file (Document No.R01AN1860EJ)
8. RZ/A1H Group Example of Initialization (for GENMAI) (Document No.R01AN1864EJ)

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

Contents

1. Overview

The USB HCDC, when used in combination with the USB-BASIC-F/W, operates as a USB host communications device class driver (HCDC). The HCDC conforms to the PSTN device subclass abstract control model of the USB communication device class specification (CDC) and enables communication with a CDC peripheral device.
This module supports the following functions.

- Checking of connected devices
- Implementation of communication line settings
- Acquisition of the communication line state
- Data transfer to and from a CDC peripheral device
- HCDC can connect maximum 2 CDC devices to 1 USB channel by using USB Hub.

1.1 Please be sure to read

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

a. The document (Document No: R01AN3291EJ) also provides how to create an application program using the APIs described above.
b. If the APIs described in the document (Document No: R01AN3291EJ) are used, there is no need to use the API described in “7.3. List of HCDC API Functions” of this document of this document.

1.2 Operation Confirmation Conditions

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

Table 1.1 Operation Confirmation Conditions

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

1.3 Limitations

This module is subject to the following restrictions

1. Structures are composed of members of different types (Depending on the compiler, the address alignment of the structure members may be shifted).

Terms and Abbreviations

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

2. Software Configuration

Table 2.1 lists the modules, and Figure 2-1 shows a block diagram of HCDC.

Table 2.1 Modules

Module	Description
APL	User application program. Created by customer.
HCDC	Requests CDC requests command and the data transfer from APL to HCD .
MGR / HUB	Enumerates the connected devices and starts HCDC. Also performs device state management.
HCD	USB host H/W control driver. (See USB Basic FW.)

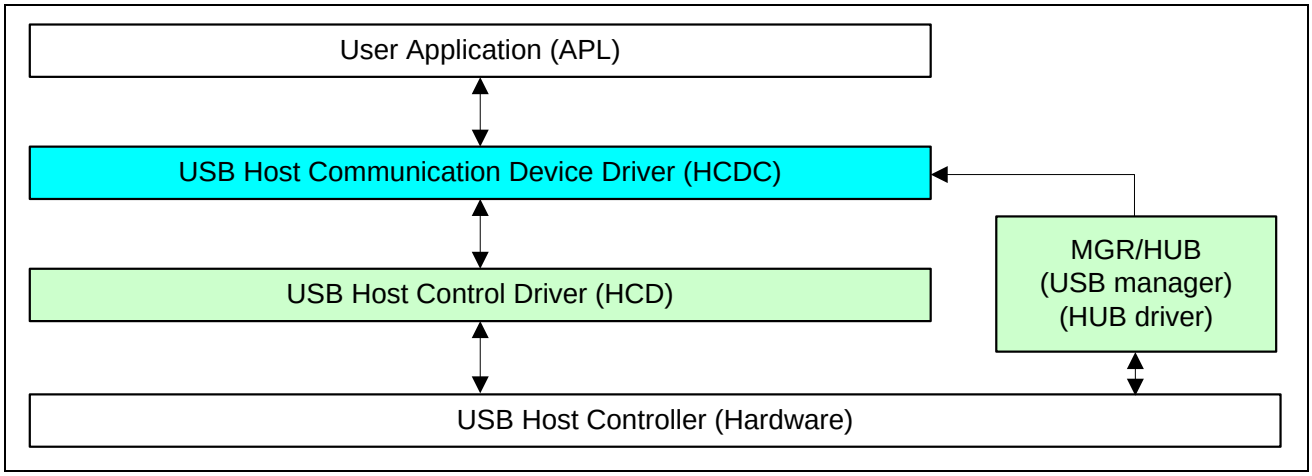

Figure 2-1 Software Block Diagram

3. System Resources

The resource which HCDC uses is showed in エラー! 参照元が見つかりません。 **Table 3.** - **Table 3.**エラー! 参照元が見つかりません。.

Table 3.1 Task Information

Function	ID	Priority	Description
usb_hcdc_Task	USB_HCDC_TSK	USB_PRI_3	HCDC Task

Table 3.2 Mailbox Information

Mailbox	ID	Queue	Description
USB_HCDC_MBX	USB_HCDC_TSK	FIFO order	for HCDC

Table 3.3 Memory Pool Information

Memory Pool	Queue	Memory block（*）	Description
USB_HCDC_MPL	FIFO order	40byte	for HCDC

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

4. Target Peripheral List（TPL）

When using a USB host driver (USB-BASIC-F/W FIT module) and device class driver in combination, it is necessary to create a target peripheral list (TPL) for each device driver.
For details, see “Target Peripheral List,” in USB Basic Firmware application note (Document No. R01AN3291EJ).

5. Compile Setting

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

Configuration options in r_usb_hcdc_config.h	
USB_HCDC_IF_CLASS	Specifies the device class ID of connected CDC devices. ・USB_IFCLS_VEN: Vendor class devices may be connected. ・USB_IFCLS_CDC: CDC class devices may be connected.
USB_HCDC_MULTI_CONNECT	Enables this definition when connecting 2 CDC devices to one USB module (USB channel).
USB_HCDC_IN_DATA_PIPE USB_HCDC_OUT_DATA_PIPE USB_HCDC_IN_DATA_PIPE2 USB_HCDC_OUT_DATA_PIPE2	Specifies the pipe number which is used at the data transfer. (Specifies any one from USB_PIPE1 to USB_PIPE5. Don't specify the same pipe number.)
USB_HCDC_STATUS_PIPE USB_HCDC_STATUS_PIPE2	Specifies the pipe number which is used at the class notification. (Specifies any one from USB_PIPE6 to USB_PIPE9. Don't specify the same pipe number.)

[Note]

1. Please confirm the specification of the CDC device before attempting to use it. When using a commercial USB-serial converter (CDC device), check that the device class ID is CDC and not Vendor class.
2. Sets the pipe number to USB_HCDC_IN_DATA_PIPE2, USB_HCDC_OUT_DATA_PIPE2 and USB_HCDC_STATUS_PIPE2 when enabling USB_HCDC_MULTI_CONNECT.

6. Communication Device Class (CDC), PSTN and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communication Device Class specification, as specified in detail in the PSTN Subclass document listed in “Related Documents”.
The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems.

6.1 Basic Functions

The main functions of HCDC are as follows.

1. Verify connected devices
2. Make communication line settings
3. Acquire the communication line state
4. Transfer data to and from the CDC peripheral device

6.2 Abstract Control Model Class Requests - Host to Device

The software supports the following ACM class requests.

Table 6.1 CDC Class Requests

Request	Code	Description
SendEncapsulatedCommand	0x00	Transmits an AT command as defined by the protocol used by the device (normally 0 for USB).
GetEncapsulatedResponse	0x01	Requests a response to a command transmitted by SendEncapsulatedCommand.
SetCommFeature	0x02	Enables or disables features such as device-specific 2-byte code and country setting.
GetCommFeature	0x03	Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting.
ClearCommFeature	0x04	Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting.
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).
GetLineCoding	0x21	Acquires the communication line setting state.
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.
SendBreak	0x23	Transmits a break signal.

For details concerning the Abstract Control Model requests, refer to Table 11, “Requests - Abstract Control Model” in “USB Communications Class Subclass Specification for PSTN Devices”, Revision 1.2.
The following describes the class request data formats supported by this class driver software.

6.2.1 SendEncapsulatedCommand

The SendEncapsulatedCommand data format is shown in Table 6.2.

Table 6.2 SendEncapsulatedCommand Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_ENCAPSULATED _COMMAND(0x00)	0x0000	0x0000	Data length	Control protocol command

Note: Items such as AT commands for modem control are set as Data, and wLength is set to match the length of the data.

6.2.2 GetEncapsulatedResponse

The GetEncapsulatedResponse data format is shown Table 6.3.

Table 6.3 GetEncapsulatedResponse Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	GET_ENCAPSULATED_ RESPONSE (0x01)	0x0000	0x0000	Data length	The data depends on the protocol.

Note: The response data to SendEncapsulatedCommand is set as Data, and wLength is set to match the length of the data.

6.2.3 SetCommFeature

The SetCommFeature data format is shown Table 6.4.

Table 6.4 SetCommFeature Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_COMM_FEATURE (0x02)	Feature Selector Note	0x0000	Data length	Status Either the country code or the Abstract Control Model idle setting/multiplexing setting for Feature Selector.

Note: Shown in Table 4.6 Feature selector Settings.

6.2.4 GetCommFeature Data Format

The GetCommFeature data format is shown below.

Table 6.5 GetCommFeature Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	GET_COMM_FEATURE (0x03)	Feature Selector Note	0x0000	Data length	Status Either the country code or the Abstract Control Model idle setting/multiplexing setting for Feature Selector.

Note: Shown in Table 4.6 Feature selector Settings.

A Feature selector setup is shown in Table 6.6. The Status format at the time of ABSTRACT_STATE is shown in Table 6.7.

Table 6.6 Feature Selector Settings

Feature Selector	Code	Targets	Length of Data	Description
RESERVED	0x00	None	None	Reserved
ABSTRACT_STATE	0x01	Interface	2	Selects the setting for Abstract Control Model idle state and signal multiplexing.
COUNTRY_SETTING	0x02	Interface	2	Selects the country code in hexadecimal format, as defined by ISO 3166.

Table 6.7 Status Format when ABSTRACT_STATE Selected

Bit Position	Description
D15 to D2	Reserved
D1	Data multiplexing setting 1: Multiplexing of call management commands is enabled for the Data class. 0: Multiplexing is disabled.
D0	Idle setting 1: No endpoints of the target interface accept data from the host, and data is not supplied to the host. 0: Endpoints continue to accept data and it is supplied to the host.

6.2.5 ClearCommFeature

The ClearCommFeature data format is shown Table 6.8.

Table 6.8 ClearCommFeature Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	CLEAR_COMM_FEATURE (0x04)	Feature Selector Note	0x0000	0x0000	None

Note: Shown in Table 4.6 Feature selector Settings.

6.2.6 SetLineCoding

The SetLineCoding data format is shown Table 6.9.

Table 6.9 SetLineCoding Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x0000	0x0000	0x0000	Line Coding Structure See Table 6.10 Line Coding Structure Format

Line Coding Structure Format is shown Table 6.10.

Table 6.10 Line Coding Structure Format

Offset	Field	Size	Value	Description
0	dwDTERate	4	Number	Data terminal speed (bps)
4	bCharFormat	1	Number	Stop bits 0 - 1 stop bit 1 - 1.5 stop bits 2 - 2 stop bits
5	bParityType	1	Number	Parity 0 - None 1 - Odd 2 - Even 3 - Mask 4 - Space
6	bDataBits	1	Number	Data bits (5, 6, 7, 8)

6.2.7 GetLineCoding

The GetLineCoding data format is shown Table 6.11.

Table 6.11 GetLineCoding Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	Line Coding Structure See Table 4.10, Line Coding Structure Format

6.2.8 SetControlLineState

The SetControlLineState data format is shown below.

Table 6.12 SetControlLineState Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_CONTROL_LINE _STATE (0x22)	Control Signal Bitmap See Table 6.13 Control Signal Bitmap	0x0000	0x0000	None

Table 6.13 Control Signal Bitmap

Bit Position	Description
D15 to D2	Reserved
D1	DCE transmit function control 0 - RTS OFF 1 - RTS ON
D0	Notification of DTE ready state 0 - DTR OFF 1 - DTR ON

6.2.9 SendBreak

The SendBreak data format is shown below.

Table 6.14 SendBreak Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_BREAK (0x23)	Break signal output duration	0x0000	0x0000	None

6.3 ACM Notifications from Device to Host

The class notifications supported and not supported by the software are shown Table 6.15.

Table 6.15 CDC Class Notifications

Notification	Code	Description	Supported
NETWORK_CONNECTION	0x00	Notification of network connection state	No
RESPONSE_AVAILABLE	0x01	Response to GET_ENCAPSLATED_RESPONSE	Yes
SERIAL_STATE	0x20	Notification of serial line state	Yes

6.3.1 SerialState

The SerialState data format is shown below.

Table 6.16 SerialState Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0000	UART State bitmap See Table 6.17 UART State bitmap Format

UART State bitmap format is shown Table 6.17.

Table 6.17 UART State bitmap Format

Bits	Field	Description
D15 to D7		Reserved
D6	bOverRun	Overrun error detected
D5	bParity	Parity error detected
D4	bFraming	Framing error detected
D3	bRingSignal	INCOMING signal (ring signal) detected
D2	bBreak	Break signal detected
D1	bTxCarrier	Data Set Ready: Line connected and ready for communication
D0	bRxCarrier	Data Carrier Detect: Carrier detected on line

6.3.2 ResponseAvailable

The ResponseAvailable data format is shown below.

Table 6.18 ResponseAvailable Data Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	RESPONSE_AVAILABLE (0x01)	0x0000	0x0000	0x0000	None

7. USB Host Communication Device Class Driver (HCDC)

7.1 Basic Functions

This software conforms to the Abstract Control Model subclass of the communication device class specification.
The main functions of HCDC are to:

1. Send class requests to the CDC peripheral
2. Transfer data to and from the CDC peripheral
3. Receive communication error information from the CDC peripheral

7.2 Structures

7.2.1 HCDC Request Structure

Table 7.1 describes the “UART settings” parameter structure used for the CDC requests *SetLineCoding* and *GetLineCoding*.

Table 7.1 USB_HCDC_LineCoding_t Structure

Type	Member	Description	Remarks
uint32_t	dwDTERate	Line speed	Unit: bps
uint8_t	bCharFormat	Stop bits setting	
uint8_t	bParityType	Parity setting	
uint8_t	bDataBits	Data bit length	

Table 7.2 describes the “UART settings” parameter structure used for the CDC requests *SetControlLineState*.

Table 7.2 USB_HCDC_ControlLineState_t Structure

Type	Member	Description	Remarks
uint16_t (D1)	bRTS:1	Carrier control for half duplex modems 0 - Deactivate carrier, 1 - Activate carrier	
uint16_t (D0)	bDTR:1	Indicates to DCE if DTE is present or not 0 - Not Present, 1 - Present	

Table 7.3 describes the “AT command” parameter structure used for the CDC requests *SendEncapsulatedCommand* and *GetEncapsulatedResponse*.

Table 7.3 USB_HCDC_Encapsulated_t Structure

Type	Member	Description	Remarks
uint8_t	*p_data	Area where AT command data is stored	
uint16_t	wLength	Size of AT command data	Unit: byte

Table 7.4 describes the “Break signal” parameter structure used for the CDC requests *SendBreak*.

Table 7.4 USB_HCDC_BreakDuration_t Structure

Type	Member	Description	Remarks
uint16_t	wTime_ms	Duration of Break	Unit: ms

7.2.2 CommFeature Function Selection Union

Table 7.5 and Table 7.6 describe the “Feature Selector” parameter structure used for the CDC requests *SetCommFeature* and *GetCommFeature*, and Table 7.7 describes the parameter union.

Table 7.5 USB_HCDC_AbstractState_t Structure

Type	Member	Description	Remarks
uint16_t	rsv1:8	Reserved1	
uint16_t	rsv2:6	Reserved2	
uint16_t	bDMS:1	Data Multiplexed State	
iomt16_t	bIS:1	Idle Setting	

Table 7.6 USB_HCDC_CountrySetting_t Structure

Type	Member	Description	Remarks
uint16_t	country_code	Country code in hexadecimal format as defined in [ISO3166],	

Table 7.7 USB_HCDC_CommFeature_t Union

Type	Member	Description	Remarks
USB_HCDC_AbstractState_t	abstractState	Abstract Control Model select time parameters	
USB_HCDC_CountrySetting_t	countrySetting	Country Setting select time parameters	

7.2.3 CDC Request Input Parameter Union

Table 7.8 describes the common parameter structure for CDC requests.

Table 7.8 USB_HCDC_ClassRequestParm_t Structure

Request	Request code Structure type	Member name	Description
SetLineCoding	USB_HCDC_SET_LINE_CODING / USB_HCDC_LineCoding_t	*LineCoding	Data address send and receive in data stage. Refer to Table 7.1
GetLineCoding	USB_HCDC_GET_LINE_CODING / USB_HCDC_LineCoding_t		
SetControlState	USB_HCDC_SET_ CONTROL_LINE_STATE / USB_HCDC_ControlLineState_t	ControlLineState	Value set to the wValue field. Refer to Table 7.2
SendEncapsulated Command	USB_HCDC_SEND_ ENCAPSULATED_COMMAND / USB_HCDC_Encapsulated_t	Encapsulated	Data address send and receive in data stage, and value set to the wValue field. Refer to Table 7.3
GetEncapsulated Response	USB_HCDC_GET_ ENACAPSULATED_RESPONSE / USB_HCDC_Encapsulated_t		
SendBreak	USB_HCDC_SEND_BREAK / USB_HCDC_BreakDuration_t	BreakDuration	Value set to the wValue field. Refer to Table 7.4
SetCommFeature	USB_HCDC_SET_ COMM_FEATURE / USB_HCDC_CommFeature_t	*CommFeature	Data address send and receive in data stage. Refer to Table 7.7
GetCommFeature	USB_HCDC_GET_ COMM_FEATURE / USB_HCDC_CommFeature_t		
ClearCommFeature	USB_HCDC_CLR_COMM_FEATURE No structure		

7.2.4 CDC Request API Function Structure

Table 7.9 describes the CDC request parameter structure.

Table 7.9 USB_HCDC_ClassRequest_UTR_t Structure

Type	Member	Description
usb_addr_t	devadr	Device address
USB_REGADR_t	ipp	USB IP Base Address
uint16_t	ip	USB IP Number
uint8_t	bRequestCode	Class request code. Refer to Table 7.8
USB_CDC_ClassRequestParm_t	parm	Parameter setup value. Refer to Table 7.8
usb_cb_t	complete	Class request processing end call-back function

7.2.5 CDC Notification Format

Table 7.10 and Table 7.11 describe the data format of the CDC notification.

Table 7.10 Response_Available notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	RESPONSE_AVAILABLE(0x01)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0000	
uint8_t	Data	none	

Table 7.11 Serial_State notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	SERIAL_STATE(0x20)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0002	
uint16_t	Data	UART State bitmap	Refer to Table 7.12

The host is notified of the "*SerialState*" when a change in the UART port state is detected. Table 7.12 describes the structure of the UART State bitmap.

Table 7.12 USB_HCDC_SerialState_t Structure

Type	Member	Description	Remarks
uint16_t (D15-D8)	rsv1:8	Reserved1	
uint16_t (D7)	rsv2:1	Reserved2	
uint16_t (D6)	bOverRun:1	Overrun error detected	
uint16_t (D5)	bParity:1	Parity error detected	
uint16_t (D4)	bFraming:1	Framing error detected	
uint16_t (D3)	bRingSignal:1	Incoming signal (Ring signal) detected	
uint16_t (D2)	bBreak:1	Break signal detected	
uint16_t (D1)	bTxCarrier:1	Line connected and ready for communication	Data Set Ready
uint16_t (D0)	bRxCarrier:1	Carrier detected on line	Data Carrier Detect

7.3 List of HCDC API Functions

The HCDC API is shown in Table 7.13.

Table 7.13 List of HCDC API Functions

Function	Description
R_usb_hcdc_receive_data	USB receive processing
R_usb_hcdc_send_data	USB send processing
R_usb_hcdc_serial_state_trans	Class notification Serial State processing
R_usb_hcdc_class_check	Descriptor check processing
R_usb_hcdc_SetPipeRegistration	Pipe setting processing
R_usb_hcdc_class_request	Sends CDC class request
R_usb_hcdc_driver_start	Driver task start setting for HCDC
R_usb_hcdc_Task	HCDC task

7.3.1 R_usb_hcdc_receive_data

Host receive data.

Format

```
USB_ER_t        R_usb_hcdc_send_data (USB_UTR_t *ptr,
                                      uint8_t *buf,
                                      uint32_t size,
                                      USB_CB_t complete)
```

Argument

ptr	Pointer to the USB Communication Structure used for attached device.
buf	Pointer to transmit data buffer address
size	Transfer size
complete	Process completion notice callback function

Return Value

–	Error code (USB_E_OK / USB_E_ERROR).

Description

This function requests USB data reception from the USB driver (HCD).
When data reception ends (specified data size reached, short packet received, error occurred), the call-back function is called. Information on remaining receive data (length, status, error count and transfer end) is determined bythe parameters of the call-back.
USB receive data is stored in the area given by the address specified by 2nd argument (**buf*).

Note

1. Call this API in the user application program or the class driver.
2. Set the following members of the *USB_UTR_t* structure when calling the function.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. Specify the area other than the auto variable (stack) area to the 2nd argument.
4. When the received data is n times of the maximum packet size and less than the specified size in the argument (*size*), it is considered that the data transfer is not ended and a callback function (*complete*) is not generated.
5. Set the device address of CDC device which do the USB data transfer to the member "*keyword*" in *USB_UTR_t* structure when the definition "USB_HCDC_MULTI_CONNECT" is enabled.
6. The USB transmit process results are obtained from the *USB_UTR_t* * argument in the call-back function
7. Refer to the structure for USB communication (USB_UTR_t structure) of a USB Basic Firmware application note.

Example

```
{
  USB_UTR_t  *ptr;
  uint16_t   size = 64;                          /* Data size */

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_HOST_USBIP_NUM;                  /* USB IP number set */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP base address set */

    R_usb_hcdc_receive_data(ptr, (uint8_t *)receive_data, size,
  (USB_CB_t)&usb_complete)
}

/* Callback function */
void  usb_complete( USB_UTR_t *mess, uint16_t data1, uint16_t data2 );
{
  /*  Describe the processing performed when the USB receive is completed.  */
}
```

7.3.2 R_usb_hcdc_send_data

Host send data

Format

```
USB_ER_t        R_usb_hcdc_send_data (USB_UTR_t *ptr,
                                      uint8_t *buf,
                                      uint32_t size,
                                      USB_CB_t complete )
```

Argument

ptr	Pointer to the USB Communication Structure used for attached device.
buf	Pointer to Transmit data buffer address
size	Transfer size
complete	Process completion notice callback function

Return Value

–	Error code (USB_E_OK / USB_E_ERROR)

Description

This function transfers the USB data in the specified transmit size from the address specified in the Transmit Data Address Table.
When the transmission processing is complete, the call-back function is called.

Note

1. Call this API in the user application program or the class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. Specify the area other than the auto variable (stack) area to the 2nd argument.
4. Set the device address of CDC device which do the USB data transfer to the member “*keyword*” in *USB_UTR_t* structure when the definition “USB_HCDC_MULTI_CONNECT” is enabled.
5. The USB transmit processing results are obtained by ”USB_UTR_t *” argument in the call-back function
6. Refer to the USB Basic Firmware application note for info on the USB communication (*USB_UTR_t)* structure.

Example

```
{
  USB_UTR_t  *ptr;

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_PERI_USBIP_NUM;                  /* USB IP number set */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP base address set */

  R_usb_hcdc_send_data(ptr, send_data, size, (USB_CB_t)&usb_complete)
}

/* Callback function */
void  usb_complete( USB_UTR_t *mess, uint16_t data1, uint16_t data2 );
{
  /*  Describe the processing performed when the USB transmit is completed.  */
}
```

7.3.3 R_usb_hcdc_serial_state_trans

Handle CDC class and serial state info from peripheral

Format

USB_ER_t R_usb_hcdc_serial_state_trans (USB_UTR_t *ptr,
USB_HCDC_SERIAL_ST_CB_t *complete)

Argument

*ptr	Pointer to the USB Communication Structure used for attached device.
complete	Process completion notice callback function

Return Value

–	Error code (USB_E_OK / USB_E_ERROR).

Description

This function receives the CDC class notification(Serial State) from the peripheral device.
Callback function complete is called after the completion of reception.
The serial status is received when the callback function is triggered.

Note

1. Call this API in the user application program or the class driver.
2. For information concerning the serial status bit pattern, refer to”Table 6.17 UART State bitmap Format.
3. The USB transmit results are obtained from the *USB_UTR_t** argument in the call-back function.
4. Please set the following member of USB_UTR_t structure when calling the function.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
5. Set the device address of CDC device which do the USB serial state reception to the member “*keyword*” in *USB_UTR_t* structure when the definition “USB_HCDC_MULTI_CONNECT” is enabled.

Example

```
void usb_hcdc_main_task(USB_VP_INT stacd)
{
  USB_UTR_t           *mess;
  USB_ER_t            err;

  while (1)
  {
  err  = R_USB_RCV_MSG(USB_HCDCSMP_MBX,(USB_MSG_t**)&mess);
  if (err == USB_OK)
  {
    err = R_usb_hcdc_serial_state_trans( mess,
          (USB_HCDC_SERIAL_ST_CB_t *)&usb_hcdc_smp_SerialStateReceive );
    if( err != USB_OK )
    {
       USB_PRINTF0("### usb_pcdc_MainTask function bulk read error\n");
    }
  }
}
```

7.3.4 R_usb_hcdc_class_check

Check descriptor

Format

void R_usb_hcdc_class_check (USB_UTR_t *ptr, uint16_t **devinfo)

Argument

*ptr Pointer to the USB Communication Structure used for attached device.
**devinfo Device information array
[0] : Device Descriptor
[1] : Configuration Descriptor
[2] : Interface Descriptor
[3] : Descriptor Check Result
[4] : HUB Classification
[5] : Port Number
[6] : Transmission Speed
[7] : Device Address

Return Value

— Result (USB_E_OK / USB_E_ERROR).

Description

This is a class driver registration function. It is registered to the driver registration structure member *classcheck*, as a callback function during HCDC registration at startup and called when a configuration descriptor is received during enumeration.
When the check result is OK, the function sets USB_DONE in the descriptor result (table[3]). When the check result is NG, the function sets USB_ERROR and ends the process.
This function references the endpoint descriptor in the peripheral device configuration descriptor, then edits the pipe information table and checks the pipe information of the pipes to be used.

Note

—

Example

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

  driver.ifclass      = (uint16_t)USB_IFCLS_CDCC;
       :
  driver.classcheck   = (USB_CB_CHECK_t)&R_usb_hcdc_class_check;
       :
  driver.devresume = (USB_CB_INFO_t)&usb_hcdc_dummy_function;
  R_usb_hstd_DriverRegistration(ptr, (USB_HCDREG_t*)&driver);
}
```

7.3.5 R_usb_hcdc_SetPipeRegistration

Set host USB H/W pipe configuration

Format

USB_ER_t R_usb_hcdc_SetPipeRegistration (USB_UTR_t *ptr, uint16_t dev_addr)

Argument

*ptr	Pointer to the USB Communication Structure used for attached device.
dev_addr	Device Address

Return Value

—	Rresult (USB_E_OK / USB_E_ERROR).

Description

This function sets the USB hardware to use the communication pipes that correspond to the endpoints used for USB CDC communications. A total of three pipes are setup in host CDC: Bulk IN and Bulk OUT pipes for data communications, as well as an Interrupt IN pipe for receiving the serial state.

Note

1. Call this API from the user application program or the class driver.
2. Please set the following member of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number

Example

```
void usb_hcdc_smp_open(USB_UTR_t *ptr, uint16_t devadr, uint16_t data2)
{
  USB_ER_t      err;

  if (devadr != 0)
  {
    usb_shcdc_devadr = devadr;     /* Device Address store */

    /* Host CDC Pipe Registration */
    err = R_usb_hcdc_SetPipeRegistration(ptr, usb_shcdc_devadr);
    if (err != USB_OK)
    {
       USB_PRINTF0("Pipe Registration error !\n");
    }
  }
}
```

7.3.6 R_usb_hcdc_class_request

Send a CDC Class request

Format

USB_ER_t R_usb_hcdc_class_request (void *pram)

Argument

*pram Class request parameter.

Return Value

– Error code (USB_E_OK / USB_E_ERROR).

Description

The following CDC class requests can be sent to an enumerated USB CDC peripheral by HCDC.

1. SendEncapsulatedCommand
2. GetEncapsulatedResponse
3. SetCommFeature
4. GetCommFeature
5. ClearCommFeature
6. SetLineCoding
7. GetLineCoding
8. SetControlLineState
9. SendBreak

Please refer to the following “Example” for details on how to issue these requests.
The parameter set in the void * pram argument will cast USB_HCDC_ClassRequestParm_t*.
Refer to Table 7.8 for the *USB_HCDC_ClassRequest_Parm* structure.

Note

1. Call this API in the user application program or the class driver.
2. Set the following members of the *USB_UTR_t* structure when calling the function.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number
3. Refer to the USB Basic Firmware application note for the USB Communication structure *USB_UTR_t*.

Example

- SetEncapsulatedResponse

```
{
    USB_ER_t    err;
    USB_HCDC_ClassRequest_UTR_t     utr_req;

    utr_req.parm.Encapsulated.p_data = p_data; /* Command data buffer */
    utr_req.parm.Encapsulated.wLength = length;
    utr_req.bRequestCode = USB_HCDC_SEND_ENCAPSULATED_COMMAND;
    utr_req.complete = smp_sendencapsulateresponse_cb;
    utr_req.devadr = devadr;                    /* Device Address */
    utr_req.ip = USB_USBIP_0;                   /* USB IP No (0/1) */
    utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );    /* USB IP address */

    /* CDC class request */
    err = R_usb_hcdc_class_request( (void*)&utr_req);

    return err;
}
/* Callback function */
void smp_sendencapsulateresponse_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
 /* Describe the processing performed when the class request is completed. */
}
```

- GetEncapsulatedResponse

```
{
 USB_HCDC_ClassRequest_UTR_t    utr_req; /* Line Coding Parameter */
 usb_er_t     err;

 /* Example of usage. */
USB_ER_t   err;
USB_HCDC_ClassRequest_UTR_t    utr_req;

utr_req.parm.Encapsulated.p_data = p_data; /* Command data buffer */
utr_req.parm.Encapsulated.wLength = length;
utr_req.bRequestCode = USB_HCDC_GET_ENACAPSULATED_RESPONSE;
utr_req.complete = smp_getencapsulateresponse_cb; /* Callback function */
utr_req.devadr = devadr;      /* USB device address */
utr_req.ip = USB_USBIP_0;      /* USB IP No (0/1)*/
utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );        /* USB IP address */

/* CDC class request */
 err = R_usb_hcdc_class_request( (void*)&utr_req);
 return err;
}
/* Callback function */
void smp_getencapsulateresponse_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
 /* Describe the processing performed when the class request is completed. */
}
```

- ***SetCommFeature***

```
{
    USB_HCDC_ClassRequest_UTR_t     utr_req;

    utr_req.bRequestCode = USB_HCDC_SET_COMM_FEATURE;
    utr_req.selector = selector; /* Feature Selector */
    utr_req.parm.CommFeature = p_commfeature; /* Feature Parameter set data */
```

```
    if( selector == USB_HCDC_ABSTRACT_STATE )
    {
       p_commfeature->abstractState.rsv = 0;
    }
     utr_req.complete = (USB_CB_t)&smp_setcommfeature_cb;
     utr_req.devadr = devadr;      /* Device Address */
     utr_req.ip = USB_USBIP_0;     /* USB IP No */
     utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

     /* CDC class request */
     err = R_usb_hcdc_class_request( (void*)&utr_req);/

     return err;
}
```

```
/* Callback function for sending SetLineCoding class request */
void smp_setcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
/* Describe the processing performed when the class request is completed. */
}
```

- ***GetCommFeature***

```
{
     USB_ER_t    err;
     USB_HCDC_ClassRequest_UTR_t     utr_req;

     utr_req.bRequestCode = USB_HCDC_GET_COMM_FEATURE;
    /* Feature Parameter storage address */
     utr_req.parm.CommFeature = p_commfeature;
     utr_req.complete = (USB_CB_t)& smp_getcommfeature_cb;
     utr_req.devadr = devadr;      /* Device Address */
     utr_req.selector = selector;  /* Feature Selector */
     utr_req.ip = USB_USBIP_0;     /* USB IP No */
     utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

     /* CDC class request */
     err = R_usb_hcdc_class_request( (void*)&utr_req);

     return err;
}
```

```
/* Callback function for sending SetLineCoding class request */
void smp_getcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
/* Describe the processing performed when the class request is completed. */
}
```

- ***ClearCommFeature***

```
{
   USB_ER_t    err;
   USB_HCDC_ClassRequest_UTR_t     utr_req;

   utr_req.bRequestCode = USB_HCDC_CLR_COMM_FEATURE;
   utr_req.complete = (USB_CB_t)&smp_clrcommfeature_cb;
   utr_req.devadr = devadr;          /* Device Address */
   utr_req.selector = selector;      /* Feature Selector */
   utr_req.ip = USB_USBIP_0;         /* USB IP No */
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

   /* CDC class request */
   err = R_usb_hcdc_class_request( (void*)&utr_req);
```

```
    return err;
  }
  /* Callback function */
  void smp_clrcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
  {
  /* Describe the processing performed when the class request is completed. */
  }
```

- ***SetLineCoding***

```
  static   USB_HCDC_LineCoding_t    usb_shcdc_line_coding;

{
   USB_ER_t   err;
   USB_HCDC_ClassRequest_UTR_t     utr_req; /* Line Coding Parameter */

   usb_shcdc_line_coding.dwDTERate    = USB_HCDC_SPEED_9600;
   usb_shcdc_line_coding.bDataBits    = USB_HCDC_DATA_BIT_8;
   usb_shcdc_line_coding.bCharFormat  = USB_HCDC_STOP_BIT_1;
   usb_shcdc_line_coding.bParityType  = USB_HCDC_PARITY_BIT_NONE;

   utr_req.bRequestCode = USB_HCDC_SET_LINE_CODING;
   utr_req.complete = (USB_CB_t)&smp_setlinecoding_cb;
   utr_req.parm.LineCoding = &usb_shcdc_line_coding;
   utr_req.devadr = devadr;
   utr_req.ip = USB_USBIP_0;    /* USB IP No */;
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

   /* CDC Class Request */
   err = R_usb_hcdc_class_request( (void*)&utr_req );
   return err;
}
  /* Callback function */
  void smp_setlinecoding_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
  {
  /* Describe the processing performed when the class request is completed. */
  }
```

- ***GetLineCoding***

```
{
      USB_ER_t    err;
      USB_HCDC_ClassRequest_UTR_t     utr_req;

      utr_req.bRequestCode = USB_HCDC_GET_LINE_CODING;
      utr_req.parm.LineCoding = p_linecoding;  /* Line Coding table address */
      utr_req.complete = smp_getlinecoding_cb;
      utr_req.devadr = devadr;       /* Device Address */
      utr_req.ip = USB_USBIP_0;    /* USB IP No */
      utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );  /* USB IP address */

      /* CDC class request */
      err = R_usb_hcdc_class_request( (void*)&utr_req);

      return err;
}
  /* Callback function for sending SetLineCoding class request  */
  void smp_getlinecoding_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
  {
        /* Describe the processing performed when the class request is completed. */
```

```
}
```

- ***SetControlLineState***

```
{
    USB_ER_t    err;
    USB_HCDC_ClassRequest_UTR_t     utr_req;

    utr_req.bRequestCode = USB_HCDC_SET_CONTROL_LINE_STATE;
    utr_req.parm.ControlLineState.bDTR = dtr;   /* RS232 signal DTR */
    utr_req.parm.ControlLineState.bRTS = rts;   /* RS232 signal RTS */
    utr_req.complete = (USB_CB_t)smp_setcontrollinestate_cb;
    utr_req.devadr = devadr;      /* Device Address */
    utr_req.ip = USB_USBIP_0; /* USB IP No */
    utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );    /* USB IP address */

    /* CDC class request */
    err = R_usb_hcdc_class_request( (void*)&utr_req);

    return err;
}
/* Callback function */
void smp_setcontrollinestate_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
/* Describe the processing performed when the class request is completed. */
}
```

- ***SendBreak***

```
{
    USB_ER_t    err;
    USB_HCDC_ClassRequest_UTR_t     utr_req;

    utr_req.bRequestCode = USB_HCDC_SEND_BREAK;
    /* Break Signal output time */
    utr_req.parm.BreakDuration.wTime_ms = time_ms;
    utr_req.complete = (USB_CB_t)smp_sendbreak_cb;
    utr_req.devadr = devadr;                /* Device Address */
    utr_req.ip = USB_USBIP_0;               /* USB IP No */
    utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );    /* USB IP address */

    /* CDC class request */
    err = R_usb_hcdc_class_request( (void*)&utr_req);

    return err;
}
/* Callback function */
void smp_sendbreak_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
/* Describe the processing performed when the class request is completed. */
}
```

7.3.7 R_usb_hcdc_driver_start

HCDC driver task init

Format

void	usb_hcdc_driver_start (void)

Argument

—	—

Return Value

—	—

Description

This function starts the HCDC driver task.

Note

Call this API from the user application at user system initialization.

Example

```
void usb_hcdc_task_start( void )
{
:
ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );
R_usb_hstd_usbdriver_start( ptr );  /* Host USB Driver Start Setting */
usb_hcdc_registration( ptr );       /* Host Application Registration */
usb_hstd_HubRegistAll(ptr);       /* Hub registration */

R_usb_hcdc_driver_start( ptr );   /* Host Class Driver Task Start Setting */
usb_hapl_task_start( ptr );       /* Host Application Task Start Setting */
R_usb_cstd_UsbIpInit( ptr, USB_HOST_PP );  /* Initialize USB IP */
:
}
```

7.3.8 R_usb_hcdc_task

HCDC task

Format

void	R_usb_hcdc_task (USB_VP_INT_t stacd)

Argument

stacd	Task start code - *Not used*

Return Value

—	—

Description

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

Note

In non-OS operations, the function is registered to be scheduled by the scheduler.

Example

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

   if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
   {
      R_usb_hstd_HcdTask((USB_VP_INT)0);   /* HCD Task */
      R_usb_hstd_MgrTask((USB_VP_INT)0);    /* MGR Task */
      R_usb_hhub_Task((USB_VP_INT)0);       /* HUB Task */
      usb_hcdc_main_task((USB_VP_INT)0);    /* HCDC Application Task */

      R_usb_hcdc_task((USB_VP_INT)0);    /* HCDC Task */

   }
   else
   {
      /* Idle Task (sleep sample) */
      R_usb_cstd_IdleTask(0);
   }
  }
}
```

8. Sample Application

8.1 Application Specifications

The main functions of the HCDC sample application (hereafter APL) are as follows.

1. Sends receive (Bulk In transfer) requests to the CDC device and receives data.
2. Transfers received data to the CDC device by means of Bulk Out transfers (loopback).
3. Makes RTS and DTR settings by means of the class request SET_CONTROL_LINE_STATE.
4. Makes communication speed and other settings when switches on the evaluation board are operated. The communication speed and other settings are made by transmitting the class request SET_LINE_CODING to the CDC device. This class request can be used to set the communication speed, number of data bits, number of stop bits, and the parity bit.
5. Acquires the communication setting values of the CDC device by sending the class request GET_LINE_CODING to the CDC device.

8.1.1 Data Transfer Image

Figure 8-1 shows the data transfer image.

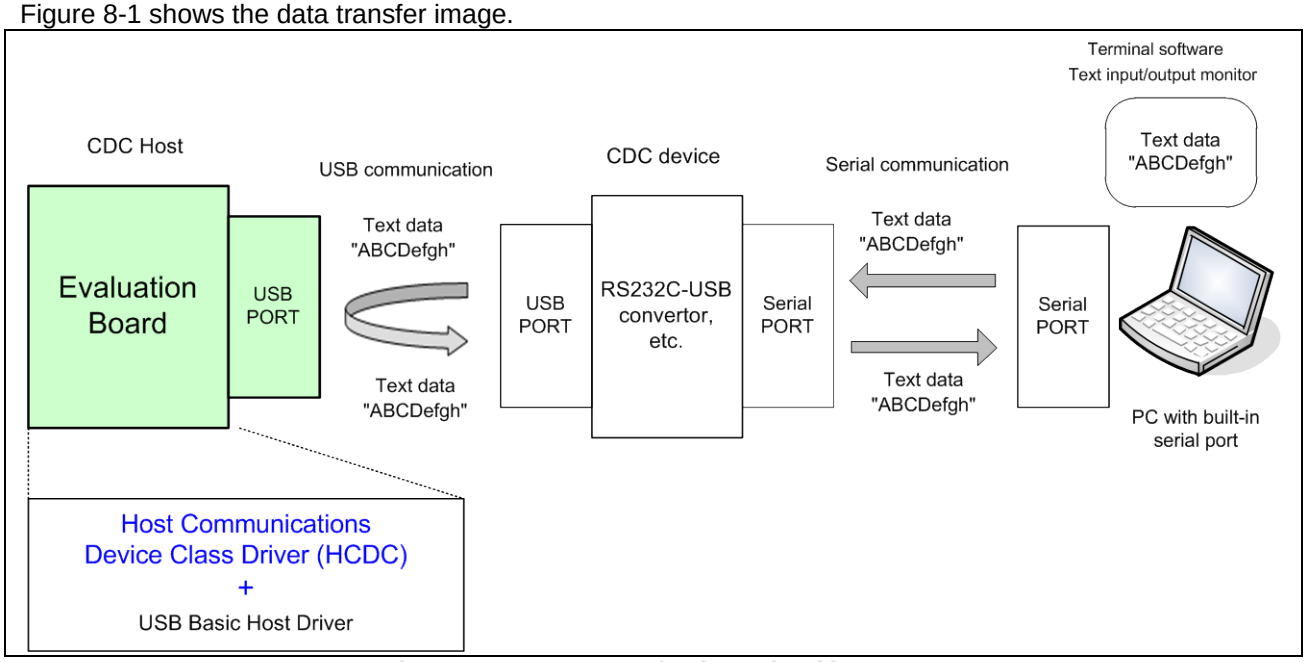

Figure 8-1 Data Transfer (Loopback) Image

8.1.2 Baud Rate Settings

The baud rate setting for the connected CDC device should match the baud rate setting of the INIT_COM_SPEED definition in the common\inc\r_usb_hcdc_apl.h file. Specify a setting of 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, or 115200 bps.

Example)

```
#define  INIT_COM_SPEED  USB_HCDC_SPEED_57600
```

8.2 Application Processing

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

8.2.1 Initial Setting

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

8.2.2 Main Loop

The main loop performs loop-back processing in which data received from the CDC device is transmitted unaltered back to the CDC device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after the CDC device attaches to the evaluation board and enumeration finishes, USB_STS_CONFIGURED is set as the return value. When the APL confirms USB_STS_CONFIGURED, it sends class request SET_LINECODING to the CDC device.
2. When it confirms that the class request processing has finished, the APL calls the R_USB_Read function to make a data receive request for data sent from the CDC device. Note that in addition to the data receive request a receive request is also sent for a class notification from the CDC device.
3. When the R_USB_GetEvent function is called after reception of data from the CDC device has finished, USB_STS_READ_COMPLETE is set as the return value. The received data is stored in external variable g_data. The receive data size can be confirmed by means of the size member of the usb_ctrl_t structure. The APL determines that a null packet has been received if the value of the size member is 0 (zero) and performs another data receive request. If the value of the size member is other than 0 (zero), the APL determines that data has been received from the CDC device. It then makes a transmit request to send the received data to the CDC device.
4. When the R_USB_GetEvent function is called after transmission of data to the CDC device finishes, USB_STS_WRITE_COMPLETE is set as the return value. When the APL confirms USB_STS_CONFIGURED, it calls the R_USB_Read function to make a data receive request for data sent by the CDC device.
5. The processing in steps 3 and 4, above, is repeated.

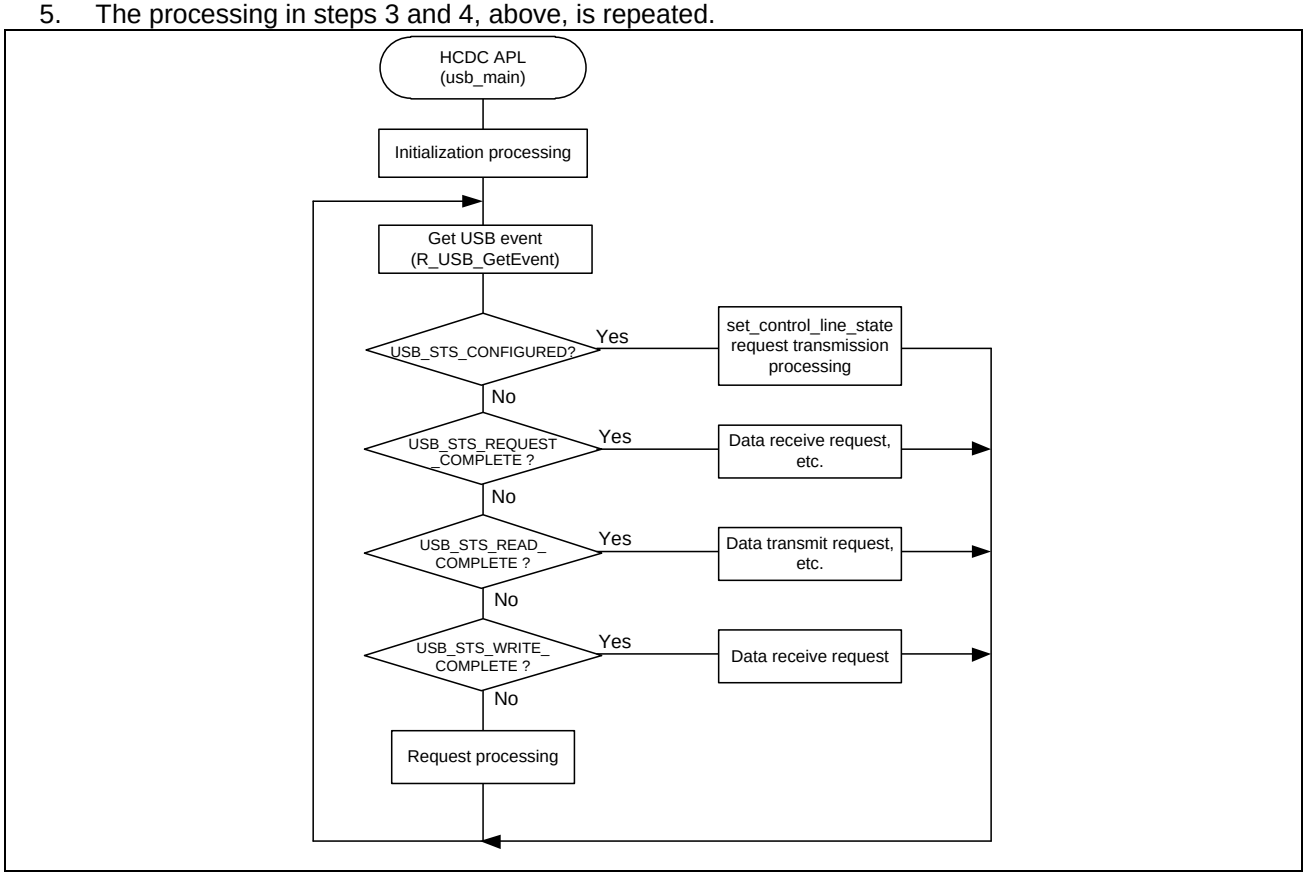

Figure 8-2 Main Loop

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 01, 2016	—	First edition issued
1.10	Sep 30, 2016	—	Since the USB-BASIC-F / W has been revised, the version up

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0