RZ/A2M Group

USB Basic Peripheral Driver

Introduction

This application note describes the USB Basic Peripheral Driver Firmware.

Target Device

RZ/A2M

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

Contents

1. Overview

The USB-BASIC-F/W performs USB hardware control. The USB-BASIC-F/W operates in combination with one type of sample device class drivers provided by Renesas.

This module supports the following functions.

<Overall>

- Supporting USB Peripheral.
- Device connect/disconnect, suspend/resume, and USB bus reset processing.
- Control transfer on pipe 0.
- Data transfer on pipes 1 to 9. (Bulk or Interrupt transfer)

<Peripheral mode>

enumeration as USB Host of USB1.1/2.0/3.0.

1.1 Note

1. This application note is not guaranteed to provide USB communication operations. The customer should verify operations when utilizing the USB device module in a system and confirm the ability to connect to a variety of different types of devices.
2. The terms "USB0 module" and "USB1 module" described in this document indicate channel 0 and channel 1 of the USB2.0 function module.

terms in this document	terms in RZ/A2M Group User's Manual: Hardware (Document number. R01UH0403746EJ)	Start address
USB0 module	USB2.0 Function Module channel 0	0xE8219000
USB1 module	USB2.0 Function Module channel 1	0xE821B000

1.2 Limitations

This driver is subject to the following limitations.

1. Multiconfigurations are not supported.
2. The USB host modes are not supported.
3. When using the USB hub for DMA transfer, only the first USB device connected to the USB hub will be able to send data using DMA transfer. All subsequent data transfers will be implemented with the CPU transfer function.
4. This USB driver does not support the error processing when the out of specification values are specified to the arguments of each function in the driver.
5. In the case of Vendor class, the user can not use the USB Hub.
6. This driver does not support the CPU transfer using D0FIFO/D1FIFO register.

1.3 Terms and Abbreviations

Abbreviation	Full Form
APL	Application program
CDP	Charging Downstream Port
DCP	Dedicated Charging Port
H/W	Renesas USB device RZ/A2M Group
MGR	Peripheral device state manager of HCD
PBC	Peripheral Battery Charging control
PCD	Peripheral control driver of USB-BASIC-F/W
PDCD	Peripheral device class driver (device driver and USB class driver)
STD	USB-BASIC-F/W
USB	Universal Serial Bus
USB-BASIC-F/W	USB Basic Peripheral firmware for RZ/A2M Group
Scheduler	Used to schedule functions, like a simplified OS.
Task	Processing unit

1.4 Software Configuration

In peripheral mode, USB-BASIC-F/W comprises the peripheral driver (PCD), and the application (APL). PDCD is the class driver and not part of the USB-BASIC-F/W. See Table 1-1.

The peripheral driver (PCD) initiate hardware control through the hardware access layer according to messages from the various tasks or interrupt handler. They also notify the appropriate task when hardware control ends, of processing results, and of hardware requests.

The customer will need to make a variety of customizations, for example designating classes, issuing vendor-specific requests, making settings with regard to the communication speed or program capacity, or making individual settings that affect the user interface.

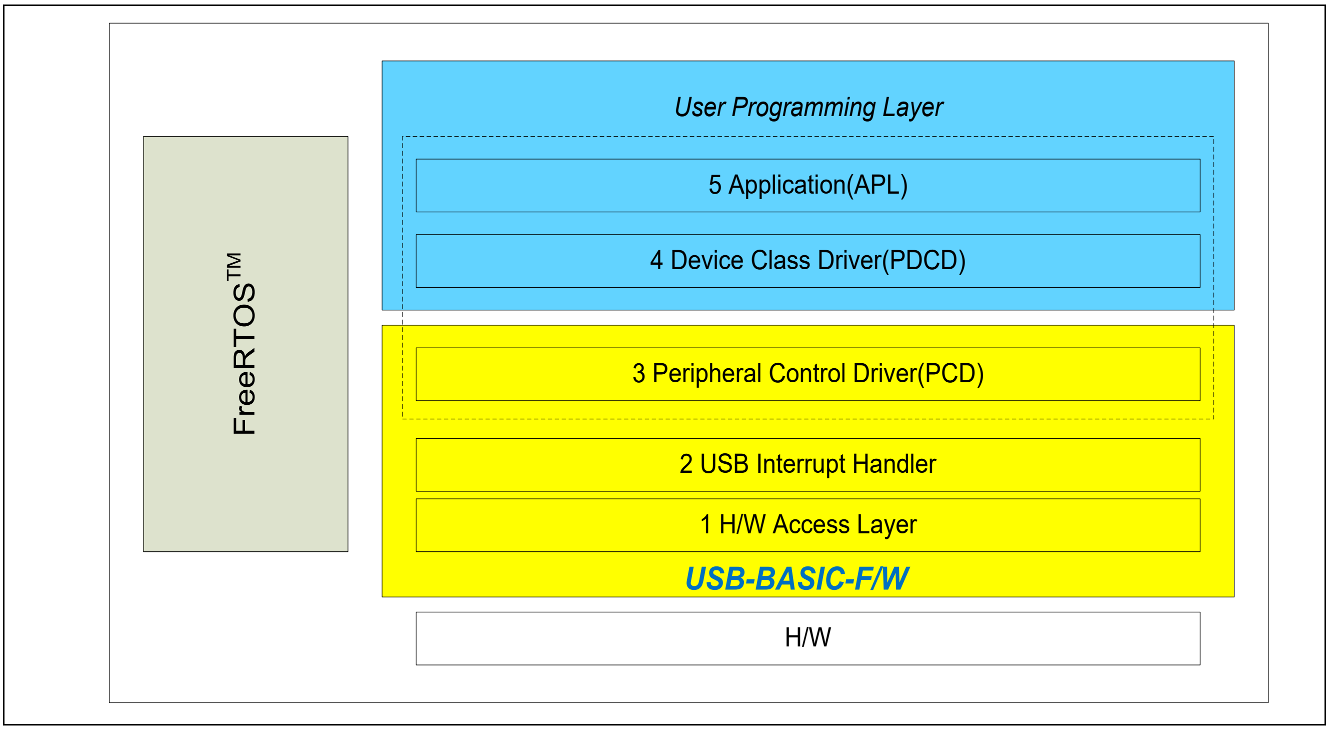

Figure 1-1 Task Configuration of USB-BASIC-F/W

Table 1-1 Software function overview

No	Module Name	Function
1	H/W Access Layer	Hardware control
2	USB Interrupt Handler	USB interrupt handler (USB packet transmit/receive end and special signal detection)
3	Peripheral Control Driver (PCD)	Hardware control in peripheral mode Peripheral transaction management
4	Device Class Driver	Provided by the customer as appropriate for the system.
5	Application	Provided by the customer as appropriate for the system.

1.5 Scheduler Function and Tasks

When using the non-OS version of the source code, a scheduler function manages requests generated by tasks and hardware according to their relative priority. When multiple task requests are generated with the same priority, they are executed using a FIFO configuration. To assure commonality with non-OS firmware, requests between tasks are implemented by transmitting and receiving messages.

1.6 Pin Setting

To use the USB module, input/output signals of the peripheral function has to be allocated to pins with the general purpose input/output port(PORT). Do the pin setting used in this module before calling R_USB_Open function.

2. Operation Confirmation Conditions

Table 2-1 Operation Confirmation Conditions(1/2)

Item	Contents
Microcomputer used	RZ/A2M
Operating frequency (Note)	CPU Clock (Iϕ) : 528MHz Image processing clock (Gϕ) : 264MHz Internal Bus Clock (Bϕ) : 132MHz Peripheral Clock 1 (P1ϕ) : 66MHz Peripheral Clock 0 (P0ϕ) : 33MHz QSPI0_SPCLK : 66MHz CKIO : 132MHz
Operating voltage	Power supply voltage (I/O): 3.3 V Power supply voltage (either 1.8V or 3.3V I/O (PVcc SPI)) : 3.3V Power supply voltage (internal): 1.2 V
Integrated development environment	e2 studio V7.8.0
C compiler	"GNU Arm Embedded Tool chain 6.3.1" compiler options(except directory path) Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0

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

Table 2-2 Operation Confirmation Conditions(2/2)

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

3. Software Overview

3.1 Peripheral Control Driver (PCD)

3.1.1 Basic functions

PCD is a program for controlling the hardware. PCD analyzes requests from PDCD and controls the hardware accordingly. It also sends notification of control results using a user provided call-back function. PCD also analyzes requests from hardware and notifies PDCD accordingly.

PCD accomplishes the following:

1. Control transfers. (Control Read, Control Write, and control commands without data stage.)
2. Data transfers. (Bulk, interrupt) and result notification.
3. Data transfer suspensions. (All pipes.)
4. USB bus reset signal detection and reset handshake result notifications.
5. Suspend/resume detections.
6. Attach/detach detection using the VBUS interrupt.
7. Hardware control when entering and returning from the clock stopped (low-power sleep mode) state.

3.1.2 Issuing requests to PCD

API functions are used when hardware control requests are issued to the PCD and when performing data transfers.

Refer to chapter 4,API Functions for the API function.

3.1.3 USB requests

This driver supports the following standard requests.

1. GET_STATUS
2. GET_DESCRIPTOR
3. GET_CONFIGURATION
4. GET_INTERFACE
5. CLEAR_FEATURE
6. SET_FEATURE
7. SET_ADDRESS
8. SET_CONFIGURATION
9. SET_INTERFACE

This driver answers requests other than the above with a STALL response.

Note that, refer to chapter 9, USB Class Requests for the processing method when this driver receives the class request or vendor request.

3.2 API Information

This Driver API follows the Renesas API naming standards.

3.2.1 Hardware Requirements

This driver requires your MCU support the following features:

- USB

3.2.2 Software Requirements

This driver is dependent upon the following packages:

- r_dmaca_rz (using DMA transfer)

3.2.3 Operating Confirmation Environment

Table 3-1 shows the operating confirmation environment of this driver.

Table 3-1 Operation Confirmation Environment

Item	Contents
Host Environment	The operation of this USB Driver module connected to the following OSes has been confirmed. 1. Windows® 7 2. Windows® 8.1 3. Windows® 10

3.2.4 Usage of Interrupt Vector

Table 3-2 shows the interrupt vector which this driver uses.

Table 3-2 List of Usage Interrupt Vectors

Device	Contents
Port0	USBFI0 Interrupt (Vector number: 64) USBFDMA00 Interrupt (Vector number: 65) USBFDMA01 Interrupt (Vector number: 66) USBFDMAERR0 Interrupt (Vector number: 67)
Port1	USBFI1 Interrupt (Vector number: 69) USBFDMA10 Interrupt (Vector number: 70) USBFDMA11 Interrupt (Vector number: 71) USBFDMAERR1 Interrupt (Vector number: 72)

3.2.5 Header Files

All API calls and their supporting interface definitions are located in r_usb_basic_if.h.

3.2.6 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in stdint.h.

3.2.7 Compile Setting

For compile settings, refer to chapter 7, Configuration(r_usb_basic_config.h).

3.2.8 Argument

For the structure used in the argument of API function, refer to chapter 8, Structures.

3.3 API (Application Programming Interface)

For the detail of the API function, refer to chapter 4, API Functions.

3.4 Class Request

For the processing method when this driver receives the class request, refer to chapter 9, USB Class Requests.

3.5 Descriptor

3.5.1 String Descriptor

This USB driver requires each string descriptor that is constructed to be registered in the string descriptor table. The following describes how to register a string descriptor.

1. First construct each string descriptor. Then, define the variable of each string descriptor in uint8_t* type.

Example descriptor construction)

```
uint8_t smp_str_descriptor0[] =
{
    0x04,             /* Length */
    0x03,             /* Descriptor type */
    0x09, 0x04        /* Language ID */
};
uint8_t smp_str_descriptor1[] =
{
    0x10,             /* Length */
    0x03,             /* Descriptor type */
    'R', 0x00,
    'E', 0x00,
    'N', 0x00,
    'E', 0x00,
    'S', 0x00,
    'A', 0x00,
    'S', 0x00
};
uint8_t smp_str_descriptor2[] =
{
    0x12,             /* Length */
    0x03,             /* Descriptor type */
    'C', 0x00,
    'D', 0x00,
    'C', 0x00,
    '_', 0x00,
    'D', 0x00,
    'E', 0x00,
    'M', 0x00,
    'O', 0x00
};
```

2. Set the top address of each string descriptor constructed above in the string descriptor table. Define the variables of the string descriptor table as uint8_t* type.

Note:

The position set for each string descriptor in the string descriptor table is determined by the index values set in the descriptor itself (iManufacturer, iConfiguration, etc.).

For example, in the table below, the manufacturer is described in smp_str_descriptor1 and the value of iManufacturer in the device descriptor is “1”. Therefore, the top address "smp_str_descriptor1" is set at Index “1” in the string descriptor table.

```
/* String Descriptor table */
uint8_t *smp_str_table[] =
{
    smp_str_descriptor0,    /* Index: 0 */
    smp_str_descriptor1,    /* Index: 1 */
    smp_str_descriptor2,    /* Index: 2 */
};
```

3. Set the top address of the string descriptor table in the *usb_descriptor_t* structure member (p_string). Refer to chapter 8.4, usb_descriptor_t structure for more details concerning the usb_descriptor_t structure.
4. Set the number of the string descriptor which set in the string descriptor table to usb_descriptor_t structure member (num_string). In the case of the above example, the value 3 is set to the member (num_string).

3.5.2 Other Descriptors

1. Please construct the device descriptor, configuration descriptor, and qualifier descriptor based on instructions provided in the **Universal Serial Bus Revision 2.0 specification**(http://www.usb.org/developers/docs/) Each descriptor variable should be defined as uint8_t* type.
2. The top address of each descriptor should be registered in the corresponding usb_descriptor_t function member. For more details, refer to chapter 8.4, usb_descriptor_t structure.

3.6 Peripheral Battery Charging (PBC)

This driver supports PBC.

PBC is the H/W control program for the target device that operates the Charging Port Detection (CPD) defined by the USB Battery Charging Specification (Revision 1.2).

You can get the result of CPD by calling R_USB_GetInformation function. For R_USB_GetInformation function, refer to chapter 4.9.

The processing flow of PBC is shown in Figure 3-1.

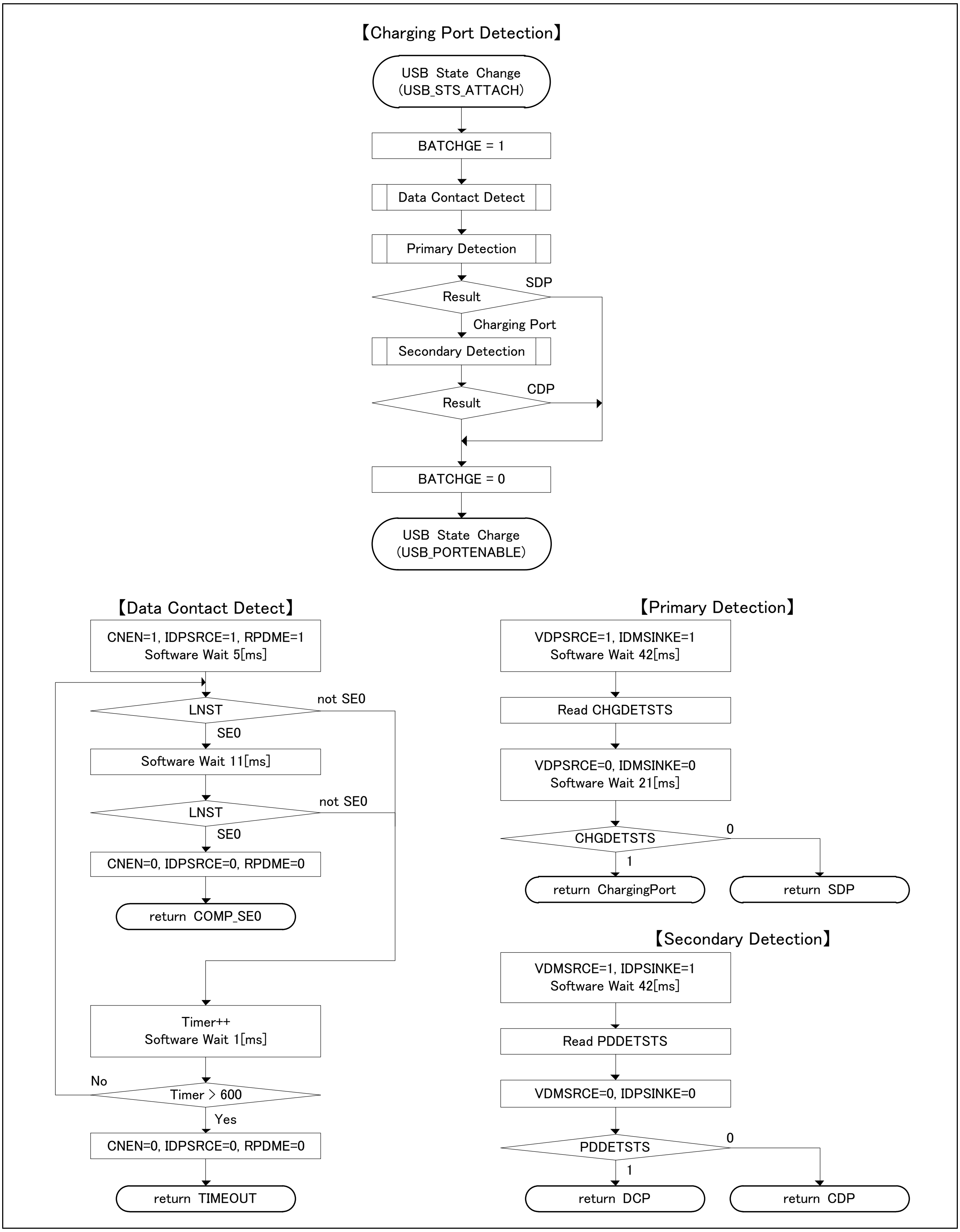

Figure 3-1 PBC processing flow

4. API Functions

Table 4-1 provides a list of API functions. These APIs can be used in common for all the classes. Use the APIs below in application programs.

Table 4-1 List of API Functions

API		Description
R_USB_Open()	(Note1)	Start the USB module
R_USB_Close()	(Note1)	Stop the USB module
R_USB_GetVersion()		Get the driver version
R_USB_Read()	(Note1)	Request USB data read
R_USB_Write()	(Note1)	Request USB data write
R_USB_Stop()	(Note1)	Stop USB data read/write processing
R_USB_Resume()	(Note1)	Request resume
R_USB_GetEvent()	(Note1)	Return USB-related completed events
R_USB_GetInformation()		Get information on USB device.
R_USB_PipeRead()	(Note1)	Request data read from specified pipe
R_USB_PipeWrite()	(Note1)	Request data write to specified pipe
R_USB_PipeStop()	(Note1)	Stop USB data read/write processing to specified pipe
R_USB_GetUsePipe()		Get pipe number
R_USB_GetPipeInfo()		Get pipe information

Note:

1. If the API of (Note 1) is executed on the same USB module by interrupt handling etc while the API of (Note 1) is executing, this USB driver may not work properly.

When USB_CFG_DISABLE is specified to USB_CFG_PARAM_CHECKING definition, the return value USB_ERR_PARA is not returned since this driver does not check the argument. Refer to chapter 7, Configuration(r_usb_basic_config.h) for USB_CFG_PARAM_CHECKING definition.

4.1 R_USB_Open

Power on the USB module and initialize the USB driver. (This is a function to be used first when using the USB module.)

Format

usb_err_t	R_USB_Open(usb_ctrl_t *p_ctrl, usb_cfg_t *p_cfg)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_cfg	Pointer to usb_cfg_t structure area

Return Value

USB_SUCCESS	Success
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Specified USB module now in use

Description

This function applies power to the USB module specified in the argument (*p_ctrl*).

Reentrant

This API is only reentrant for different USB module.

Note

1. For details concerning the usb_ctrl_t structure, see chapter 8.1, usb_ctrl_t structure, and for the usb_cfg_t structure, see chapter 8.3, usb_cfg_t structure.
2. Specify the number of the module (USB_IP0/USB_IP1) to be started up in member (module) of the usb_ctrl_t structure. Specify “USB_IP0” to start up the USB0 module and “USB_IP1” to start up the USB1 module. If something other than USB_IP0 or USB_IP1 is assigned to the member (module), then USB_ERR_PARA will be the return value.
3. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
4. Assign the device class type (see chapter 6, Device Class Types) to the member (type) of the usb_ctrl_t structure. Does not assign USB_HCDCC and USB_PCDCC to this member (type).
5. In the usb_cfg_t structure member (usb_mode), specify “USB_PERI” to start up USB peripheral operations If these settings are not supported by the USB module, USB_ERR_PARA will be returned.
6. Specify the USB speed (USB_HS / USB_FS) in the usb_ctrl_t structure member (usb_speed). If the speed set in the member is not supported by the USB module, USB_ERR_PARA will be returned.
7. Assign a pointer to the usb_descriptor_t structure to the member (p_usb_reg) of the usb_cfg_t structure. This assignment is only effective if “USB_PERI” is assigned to the member (usb_mode).

Examples

1. **In the case of USB Peripheral**

```
usb_descriptor_t smp_descriptor =
{
    g_device,
    g_config_f,
    g_config_h,
    g_qualifier,
    g_string
};
void usb_peri_application(void)
{
    usb_err_t      err;
    usb_ctrl_t     ctrl;
    usb_cfg_t      cfg;
                 :
    ctrl.module    = USB_IP1;
    ctrl.type      = USB_PCDC;
    cfg.usb_mode   = USB_PERI;
    cfg.usb_speed  = USB_HS;
    cfg.p_usb_reg  = &smp_descriptor;
    err            = R_USB_Open(&ctrl, &cfg );      /* Start USB module */
    if (USB_SUCCESS != err)
    {
                 :
    }
                 :
}
```

4.2 R_USB_Close

Power off USB module.

Format

usb_err_t	R_USB_Close(usb_ctrl_t *p_ctrl)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area

Return Value

USB_SUCCESS	Success
USB_ERR_PARA	Parameter error
USB_ERR_NOT_OPEN	USB module is not open.

Description

This function terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module).

Reentrant

This API is only reentrant for different USB module.

Note

1. Specify the number of the USB module (USB_IP0/USB_IP1) to be stopped in the usb_ctrl_t structure member (module). If something other than USB_IP0 or USB_IP1 is assigned to the member (module), then USB_ERR_PARA will be the return value.
2. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.

Example

```
void usr_application(void)
{
    usb_err_t        err;
    usb_ctrl_t       ctrl;
            :
    ctrl.module   = USB_IP0
    err           = R_USB_Close(&ctrl);
    if (USB_SUCCESS != err)
    {
            :
    }
            :
}
```

4.3 R_USB_GetVersion

Return API version number

Format

usb_err_t R_USB_GetVersion()

Arguments

— —

Return Value

Version number

Description

The version number of the USB driver is returned.

Reentrant

This API is reentrant.

Note

--

1. Example

```
void usr_application( void )
{
    uint32_t   version;
             :
    version = R_USB_GetVersion();
             :
}
```

4.4 R_USB_Read

USB data read request

Format

usb_err_t	R_USB_Read(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_buf	Pointer to area that stores read data
size	Read request size

Return Value

USB_SUCCESS	Successfully completed (Data read request completed)
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Data receive request already in process for USB device with same device address.
USB_ERR_NG	Other error

Description

1. Bulk/interrupt data transfer
   Requests USB data read (bulk/interrupt transfer).
   The read data is stored in the area specified by argument (p_buf).
   After data read is completed, confirm the operation by checking the return value (USB_STS_READ_COMPLETE) of the R_USB_GetEvent function. The received data size is set in member (size) of the usb_ctrl_t structure. To figure out the size of the data when a read is complete, check the return value (USB_STS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.
2. Control data transfer
   Refer to chapter 9, USB Class Requests for details.

Reentrant

This API is only reentrant for different USB module.

Note

1. This API only performs data read request processing. An application program does not wait for data read completion by using this API.
2. When USB_SUCCESS is returned for the return value, it only means that a data read request was performed to the USB driver, not that the data read processing has completed. The completion of the data read can be checked by reading the return value (USB_STS_READ_COMPLETE) of the R_USB_GetEvent function.
3. When the read data is n times the maximum packet size and does not meet the read request size, the USB driver assumes the data transfer is still in process and USB_STS_READ_COMPLETE is not set as the return value of the R_USB_GetEvent function.
4. Before calling this API, assign the device class type (see chapter 6, Device Class Types) to the member (type) of the usb_ctrl_t structure.
5. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
6. Do not assign a pointer to the auto variable (stack) area to the second argument (p_buf). Allocate the area of the following size when using DMA transer.
   (1). When USB_CFG_CNTMDON is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h.
   Allocate the area more than n times FIFO buffer size. For FIFO buffer size, refer to the chapter 11.3, Reference or Change of PIPEBUF Register
   (2). When USB_CFG_CNTMDOFF is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h.
   Allocate the area n times the max packet size.
7. The size of area assigned to the second argument (p_buf) must be at least as large as the size specified for the third argument (size).

8. If 0 (zero) is assigned to one of the arguments, USB_ERR_PARA will be the return value.
9. In USB Peripheral mode it is not possible to repeatedly call the R_USB_Read function with the same value assigned to the member (type) of the usb_crtl_t structure. If the R_USB_Read function is called repeatedly, then USB_ERR_BUSY will be the return value. To call the R_USB_Read function more than once with the same value assigned to the member (type), first check the USB_STS_READ_COMPLETE return value from the R_USB_GetEvent function, and then call the R_USB_Read function.
10. In Vendor Class, use the R_USB_PipeRead function.
11. If this API is called after assigning USB_PCDCC, USB_HMSC, USB_PMSC, USB_HVND or USB_PVND to the member (type) of the usb_crtl_t structure, then USB_ERR_PARA will be the return value.
12. In the USB device is in the CONFIGURED state, this API can be called. If this API is called when the USB device is in other than the CONFIGURED state, then USB_ERR_NG will be the return value.

Example

```
void usb_application( void )
{
    usb_ctrl_t    ctrl;
                  :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                  :
            case USB_STS_WRITE_COMPLETE:
                  :
                ctrl.module   = USB_IP1
                ctrl.adderss  = adr;
                ctrl.type     = USB_HCDC;
                R_USB_Read(&ctrl, g_buf, DATA_LEN);
                  :
            break;
            case USB_STS_READ_COMPLETE:
                  :
            break;
                  :
        }
    }
}
```

4.5 R_USB_Write

USB data write request

Format

usb_err_t	R_USB_Write(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_buf	Pointer to area that stores write data
size	Write size

Return Value

USB_SUCCESS	Successfully completed (Data write request completed)
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Data write request already in process for USB device with same device address.
USB_ERR_NG	Other error

Description

1. Bulk/Interrupt data transfer
   Requests USB data write (bulk/interrupt transfer).
   Stores write data in area specified by argument (p_buf).
   Set the device class type in usb_ctrl_t structure member (type).
   Confirm after data write is completed by checking the return value (USB_STS_WRITE_COMPLETE) of the R_USB_GetEvent function.
   To request the transmission of a NULL packet, assign USB_NULL(0) to the third argument (size).
2. Control data transfer
   Refer to chapter 9, USB Class Requests for details.

Reentrant

This API is only reentrant for different USB module.

Note

1. This API only performs data write request processing. An application program does not wait for data write completion by using this API.
2. When USB_SUCCESS is returned for the return value, it only means that a data write request was performed to the USB driver, not that the data write processing has completed. The completion of the data write can be checked by reading the return value (USB_STS_WRITE_COMPLETE) of the R_USB_GetEvent function.
3. Before calling this API, assign the device class type (see chapter 6, Device Class Types) to the member (type) of the usb_ctrl_t structure.
4. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
5. Do not assign a pointer to the auto variable (stack) area to the second argument (p_buf).
6. If USB_NULL is assigned to the argument (p_ctrl), then USB_ERR_PARA will be the return value.
7. If a value other than 0 (zero) is set for the argument (size) and USB_NULL is assigned to the argument (p_buf), then USB_ERR_PARA will be the return value.
8. In USB Peripheral mode it is not possible to repeatedly call the R_USB_Write function with the same value assigned to the member (type) of the usb_crtl_t structure. If the R_USB_Write function is called repeatedly, then USB_ERR_BUSY will be the return value. To call the R_USB_Write function more than once with the same value assigned to the member (type), first check the USB_STS_WRITE_COMPLETE return value from the R_USB_GetEvent function, and then call the R_USB_Write function.
9. In Vendor Class, use the R_USB_PipeWrite function.
10. If this API is called after assigning USB_HCDCC, USB_HMSC, USB_PMSC, USB_HVND or USB_PVND to the member (type) of the usb_crtl_t structure, then USB_ERR_PARA will be the return value.

11. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t    ctrl;
                  :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                      :
            case USB_STS_READ_COMPLETE:
                      :
                ctrl.module  = USB_IP0;
                ctrl.address = adr;
                ctrl.type    = USB_HCDC;
                R_USB_Write(&ctrl, g_buf, size);
                      :
            break;
            case USB_STS_WRITE_COMPLETE:
                    :
            break;
                    :
        }
    }
}
```

4.6 R_USB_Stop

USB data read/write stop request

Format

usb_err_t	R_USB_Stop(usb_ctrl_t *p_ctrl, uint16_t type)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
type	Receive (USB_READ) or send (USB_WRITE)

Return Value

USB_SUCCESS	Successfully completed (stop completed)
USB_ERR_PARA	Parameter error
USB_ERR_NG	Other error

Description

This function is used to request a data read/write transfer be terminated when a data read/write transfer is performing.

To stop a data read, set USB_READ as the argument (type); to stop a data write, specify USB_WRITE as the argument (type).

Reentrant

This API is only reentrant for different USB module.

Note

1. Before calling this API, assign the device class type to the member (type) of the usb_ctrl_t structure.
2. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
3. If USB_NULL is assigned to the argument (p_ctrl), then USB_ERR_PARA will be the return value.
4. If USB_HCDCC is assigned to the member (type) and USB_WRITE is assigned to the 2nd argment (type), then USB_ERR_PARA will be the return value.
5. If USB_PCDCC is assigned to the member (type) and USB_READ is assigned to the 2nd argment (type), then USB_ERR_PARA will be the return value.
6. If something other than USB_READ or USB_WRITE is assigned to the third argument (type), then USB_ERR_PARA will be the return value.
7. When the R_USB_GetEvent function is called after a data read/write stopping has been completed, the return value USB_STS_READ_COMPLETE/USB_STS_WRITE_COMPLETE is returned.
8. If this API is called after assigning USB_HMSC, USB_PMSC, USB_HVND or USB_PVND to the member (type) of the usb_crtl_t structure, then USB_ERR_PARA will be the return value.
9. In Vendor Class, use the R_USB_PipeStop function.
10. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t    ctrl;
                      :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                      :
            case USB_STS_DETACH:
                      :
                ctrl.module   = USB_IP1;
                ctrl.address  = adr;
                ctrl.type     = USB_HCDC;
                R_USB_Stop(&ctrl, USB_READ );   /* Receive stop */
                R_USB_Stop(&ctrl, USB_WRITE );  /* Send stop */
                      :
            break;
                      :
        }
    }
}
```

4.7 R_USB_Resume

Resume signal transmission

Format

usb_err_t	R_USB_Resume(usb_ctrl_t *p_ctrl)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_NOT_SUSPEND	USB device is not in the SUSPEND state.

Description

This function sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_t structure.

After the resume request is completed, confirm the operation with the return value (USB_STS_RESUME) of the R_USB_GetEvent function

Reentrant

This API is only reentrant for different USB module.

Note

1. This API only performs a Resume signal transmission request. An application program does not wait for Resume signal transmission completion by using this API.
2. Please call this API after calling the R_USB_Open function (and before calling the R_USB_Close function).
3. This API can be used for RemoteWakeup only with HID Class in USB Peripheral mode. In this case, the USB module number is not required to be assigned to the member (module) of the usb_ctrl_t structure. If the peripheral device class other than USB_PHID is assigned to the member (type) of the usb_ctrl_t structure, then USB_ERR_PARA will be the return value.
4. Assign the USB module to which the RESUME signal is transmitted to the member (module) of the usb_ctrl_t structure. USB_IP0 or USB_IP1 should be assigned to the member (module). If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
5. This API can be called when the USB device is in the suspend state. When the API is called in any other state, USB_ERR_NOT_SUSPEND is returned.

Example

1. **In the case of HID device(USB Peripheral)**

```
void usb_peri_application( void )
{
    usb_ctrl_t ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent( &ctrl ))
        {
                    :
            case USB_STS_NONE:
                    :
                R_USB_Resume(&ctrl);
                    :
            break;
            case USB_STS_RESUME:
                    :
            break;
                    :
         }
    }
}
```

4.8 R_USB_GetEvent

Get completed USB-related events

Format

usb_err_t R_USB_GetEvent(usb_ctrl_t *p_ctrl)

Arguments

p_ctrl Pointer to usb_ctrl_t structure area

Return Value

-- Value of completed USB-related events

Description

This function obtains completed USB-related events.

In USB peripheral mode, USB_NULL is specified in member (address).

Reentrant

This API is not reentrant.

Note

1. Please call this API after calling the R_USB_Open function (and before calling the R_USB_Close function).
2. Refer to chapter 5, Return Value of R_USB_GetEvent Function" for details on the completed event value used as the API return value.
3. If there is no completed event when calling this API, then USB_STS_NONE will be the return value.
4. Please call this API in the main loop of the user application program.

Example

```
void usb_application( void )
{
    usb_ctrl_t    ctrl;
                        :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                        :
            case USB_STS_CONFIGURED:
                        :
            break;
                        :
        }
    }
}
```

4.9 R_USB_GetInformation

Get USB device information

Format

usb_err_t	R_USB_GetInformation(usb_ctrl_t *p_ctrl, usb_info_t *p_info)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_info	Pointer to usb_info_t structure area

Return Value

USB_SUCCESS	Successful completion (VBUS supply start/stop completed)
USB_ERR_PARA	Parameter error

Description

This function obtains completed USB-related events.

For information to be obtained, see chapter 8.6, usb_info_t structure.

Reentrant

This API is reentrant.

Note

1. Call this API after calling the R_USB_Open function (and before calling the R_USB_Close function).
2. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
3. In USB Peripheral mode, assign USB_NULL to the first arugument (p_ctrl).
4. Do not assign USB_NULL to the second arugument (p_info). If USB_NULL is assigned, then USB_ERR_PARA will be the return value.

Example

1. In the case of USB Peripheral mode

```
void usb_peri_application( void )
{
    usb_ctrl_t   ctrl;
    usb_info_t   info;
              :
    R_USB_GetInformation( (usb_ctrl_t *)USB_NULL, &info );
              :
}
```

4.10 R_USB_PipeRead

Request data read via specified pipe

Format

usb_err_t	R_USB_PipeRead(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_buf	Pointer to area that stores data
size	Read request size

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Specifed pipe now handling data receive/send request
USB_ERR_NG	Other error

Description

This function requests a data read (bulk/interrupt transfer) via the pipe specified in the argument.

The read data is stored in the area specified in the argument (p_buf).

After the data read is completed, confirm the operation with the R_USB_GetEvent function return value (USB_STS_READ_COMPLETE). To figure out the size of the data when a read is complete, check the return value (USB_STS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

Reentrant

This API is reentrant for different USB PIPE

Note

1. This API only performs data read request processing. An application program does not wait for data read completion by using this API.
2. When USB_SUCCESS is returned for the return value, it only means that a data read request was performed to the USB driver, not that the data read processing has completed. The completion of the data read can be checked by reading the return value (USB_STS_READ_COMPLETE) of the R_USB_GetEvent function.
3. When the read data is n times the max packet size and does not meet the read request size, the USB driver assumes the data transfer is still in process and USB_STS_READ_COMPLETE is not set as the return value of the R_USB_GetEvent function.
4. Before calling this API, assign the PIPE number (USB_PIPE1 to USB_PIPE9) to be used to the member (pipe) of the usb_ctrl_t structure.
5. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
6. If something other than USB_PIPE1 through USB_PIPE9 is assigned to the member (pipe) of the usb_ctrl_t structure, then USB_ERR_PARA will be the return value.
7. Do not assign a pointer to the auto variable (stack) area to the second argument (p_buf).
8. The size of area assigned to the second argument (p_buf) must be at least as large as the size specified for the third argument (size). Allocate the area of the following size when using DMA transer.
   (1). When USB_CFG_CNTMDON is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h.
   Allocate the area more than n times FIFO buffer size. For FIFO buffer size, refer to the chapter 11.3, Reference or Change of PIPEBUF Register
   (2). When USB_CFG_CNTMDOFF is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h.
   Allocate the area n times the max packet size.
9. If 0 (zero) is assigned to one of the arguments, then USB_ERR_PARA will be the return value.

10. It is not possible to repeatedly call the R_USB_PipeRead function with the same value assigned to the member (pipe) of the usb_crtl_t structure. If the R_USB_PipeRead function is called repeatedly, then USB_ERR_BUSY will be the return value. To call the R_USB_PipeRead function more than once with the same value assigned to the member (pipe), first check the USB_STS_READ_COMPLETE return value from the R_USB_GetEvent function, and then call the R_USB_PipeRead function.
11. In CDC/HID Class, to perform a Bulk/Interrupt transfer, use the R_USB_Read function rather than this API.
12. Assign nothing to the member (type) of the usb_ ctrl_t structure. Even if the device class type or something is assigned to the member (type), it is ignored.
13. To transfer the data for a Control transfer, use the R_USB_Read function rather than this API.
14. Enable one of USB_CFG_HVND_USB or USB_CFG_PVND_USE definition when using this API. If this API is used when these definitions are not enabled, USB_ERR_NG is returned. For USB_CFG_HVND_USB or USB_CFG_PVND_USE definition, refer to chapter 7, Configuration(r_usb_basic_config.h).
15. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t     ctrl;
                     :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                     :
            case USB_STS_WRITE_COMPLETE:
                     :
                ctrl.module   = USB_IP1;
                ctrl.pipe     = USB_PIPE1;
                R_USB_PipeRead(&ctrl, buf, size);
                     :
            break;
            case USB_STS_READ_COMPLETE:
                     :
            break;
                     :
        }
    }
}
```

4.11 R_USB_PipeWrite

Request data write to specified pipe

Format

usb_err_t	R_USB_PipeWrite(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_buf	Pointer to area that stores data
size	Write request size

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Specifed pipe now handling data receive/send request
USB_ERR_NG	Other error

Description

This function requests a data write (bulk/interrupt transfer).

The write data is stored in the area specified in the argument (p_buf).

After data write is completed, confirm the operation with the return value (USB_STS_WRITE_COMPLETE) of the R_USB_GetEvent function.

To request the transmission of a NULL packet, assign USB_NULL (0) to the third argument (size).

Reentrant

This API is reentrant for different USB PIPE

Note

1. This API only performs data write request processing. An application program does not wait for data write completion by using this API.
2. When USB_SUCCESS is returned for the return value, it only means that a data write request was performed to the USB driver, not that the data write processing has completed. The completion of the data write can be checked by reading the return value (USB_STS_WRITE_COMPLETE) of the R_USB_GetEvent function.
3. Before calling this API, assign the PIPE number (USB_PIPE1 to USB_PIPE9) to be used to the member (pipe) of the usb_ctrl_t structure.
4. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
5. If something other than USB_PIPE1 through USB_PIPE9 is assigned to the member (pipe) of the usb_ctrl_t structure, then USB_ERR_PARA will be the return value.
6. Do not assign a pointer to the auto variable (stack) area to the second argument (p_buf).
7. If 0 (zero) is assigned to the argument (p_ctrl or p_buf), then USB_ERR_PARA will be the return value.
8. It is not possible to repeatedly call the R_USB_PipeWrite function with the same value assigned to the member (pipe) of the usb_crtl_t structure. If the R_USB_PipeWrite function is called repeatedly, then USB_ERR_BUSY will be the return value. To call the R_USB_PipeWrite function more than once with the same value assigned to the member (pipe), first check the USB_STS_WRITE_COMPLETE return value from the R_USB_GetEvent function, and then call the R_USB_PipeWrite function.
9. In CDC/HID Class, to perform a Bulk/Interrupt transfer, use the R_USB_Write function rather than this API.
10. Assign nothing to the member (type) of the usb_ ctrl_t structure. Even if the device class type or something is assigned to the member (type), it is ignored.
11. To transfer the data for a Control transfer, use the R_USB_Write function rather than this API.
12. Enable one of USB_CFG_HVND_USB or USB_CFG_PVND_USE definition when using this API. If this API is used when these definitions are not enabled, USB_ERR_NG is returned. For

USB_CFG_HVND_USB or USB_CFG_PVND_USE definition, refer to chapter 7, Configuration(r_usb_basic_config.h).

13. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t      ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_READ_COMPLETE:
                    :
                ctrl.moudle   = USB_IP0;
                ctrl.pipe     = USB_PIPE2;
                R_USB_PipeWrite(&ctrl, g_buf, size);
                    :
            break;
            case USB_STS_WRITE_COMPLETE:
                    :
            break;
                    :
        }
    }
}
```

4.12 R_USB_PipeStop

Stop data read/write via specified pipe

Format

usb_err_t	R_USB_PipeStop(usb_ctrl_t *p_ctrl)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area

Return Value

USB_SUCCESS	Successfully completed (stop request completed)
USB_ERR_PARA	Parameter error
USB_ERR_BUSY	Stop request already in process for USB device with same device address.
USB_ERR_NG	Other error

Description

This function is used to terminate a data read/write operation.

Reentrant

This API is reentrant for different USB PIPE

Note

1. Before calling this API, specify the selected pipe number (USB_PIPE0 to USB_PIPE9) in the usb_ctrl_t member (pipe). In USB Peripheral mode, no assignment to the members (address and module) is required. If assignment is performed, it is ignored.
2. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
3. If something other than USB_PIPE1 through USB_PIPE9 is assigned to the member (pipe) of the usb_ctrl_t structure, then USB_ERR_PARA will be the return value.
4. When the R_USB_GetEvent function is called after a data read/write stopping has been completed, the return value USB_STS_READ_COMPLETE/USB_STS_WRITE_COMPLETE is returned.
5. Assign nothing to the member (type) of the usb_ ctrl_t structure. Even if the device class type or something is assigned to the member (type), it is ignored.
6. Enable one of USB_CFG_HVND_USB or USB_CFG_PVND_USE definition when using this API. If this API is used when these definitions are not enabled, USB_ERR_NG is returned. For USB_CFG_HVND_USB or USB_CFG_PVND_USE definition, refer to chapter 7, Configuration(r_usb_basic_config.h).
7. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t      ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_DETACH:
                    :
                ctrl.module   = USB_IP0;
                ctrl.pipe     = USB_PIPE1;
                R_USB_PipeStop( &ctrl );
                    :
            break;
                    :
        }
    }
}
```

4.13 R_USB_GetUsePipe

Get used pipe number from bit map

Format

usb_err_t R_USB_GetUsePipe(usb_ctrl_t *p_ctrl, uint16_t *p_pipe)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_pipe	Pointer to area that stores the selected pipe number (bit map information)

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_NG	Other error

Description

Get the selected pipe number (number of the pipe that has completed initalization) via bit map information. The bit map information is stored in the area specified in argument (p_pipe). Based on the information (module member and address member) assigned to the usb_ctrl_t structure, obtains the PIPE information of that USB device.

The relationship between the pipe number specified in the bit map information and the bit position is shown below.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
--	--	--	--	--	--	PIPE9	PIPE8	PIPE7	PIPE6	PIPE5	PIPE4	PIPE3	PIPE2	PIPE1	PIPE0
0	0	0	0	0	0	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	1

0:Not used, 1: Used

For example, when PIPE1, PIPE2, and PIPE8 are used, the value “0x0107” is set in the area specified in argument (p_pipe).

Reentrant

This API is reentrant.

Note

1. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
2. In USB Peripheral mode, assign USB_NULL to the first argument (p_ctrl).
3. Bit map information b0(PIPE0) is always set to "1".
4. This API can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

1. **In the case of USB Peripheral mode**

```
void usb_application( void )
{
    uint16_t     usepipe;
    usb_ctrl_t   ctrl;

    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                        :
            case USB_STS_CONFIGURED:
                        :
                R_USB_GetUsePipe((usb_ctrl_t *)USB_NULL, &usepipe);
                        :
            break;
                        :
        }
    }
}
```

4.14 R_USB_GetPipeInfo

Get pipe information for specified pipe

Format

usb_err_t R_USB_GetPipeInfo(usb_ctrl_t *p_ctrl, usb_pipe_t *p_info)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_info	Pointer to usb_pipe_t structure area

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_NG	Other error

Description

This function gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. The obtained pipe information is stored in the area specified in the argument (p_info).

Reentrant

This API is reentrant.

Note

1. Before calling this API, specify the pipe number (USB_PIPE1 to USB_PIPE9) in the usb_ctrl_t structure member (pipe).
2. If the MCU being used only supports one USB module, then do not assign USB_IP1 to the member (module). If USB_IP1 is assigned, then USB_ERR_PARA will be the return value.
3. In USB Peripheral mode, no assignment to the members (address and module) is required.
4. Refer to chapter 8.5, usb_pipe_t structure for details on the usb_pipe_t structure.
5. This function can be called when the USB device is in the configured state. When the API is called in any other state, USB_ERR_NG is returned.

Example

```
void usb_application( void )
{
    usb_pipe_t      info;
    usb_ctrl_t      ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_CONFIGURED:
                    :
                ctrl.pipe     = USB_PIPE3;
                ctrl.module   = USB_IP1;
                ctrl.address  = address;
                R_USB_GetPipeInfo( &ctrl, &info );
                    :
            break;
                    :
        }
    }
}
```

5. Return Value of R_USB_GetEvent Function

The return values for the R_USB_GetEvent function are listed below. Make sure you describe a program in the application program to be triggered by each return value from the R_USB_GetEvent function.

Return Value	Description	Peri
USB_STS_DEFAULT	USB device has transitioned to default state.	○
USB_STS_CONFIGURED	USB device has transitioned to configured state.	○
USB_STS_SUSPEND	USB device has transitioned to suspend state.	○
USB_STS_RESUME	USB device has returned from suspend state.	○
USB_STS_DETACH	USB device has been detached from USB host.	○
USB_STS_REQUEST	USB device received USB request (Setup).	○
USB_STS_REQUEST_COMPLETE	USB request data transfer/receive is complete; device has transitioned to status stage.	○
USB_STS_READ_COMPLETE	USB data read processing is complete.	○
USB_STS_WRITE_COMPLETE	USB data write processing is complete.	○
USB_STS_NONE	No USB-related events.	○

〇:support ×:not support

5.1 USB_STS_DEFAULT

When the R_USB_GetEvent function is called after the USB device has transitioned to the default state, the function sends USB_STS_DEFAULT as the return value.

5.2 USB_STS_CONFIGURED

When the R_USB_GetEvent function is called after the USB device has transitioned to the configured state, the function sends USB_STS_CONFIGURED as the return value.

5.3 USB_STS_SUSPEND

When the R_USB_GetEvent function is called after the USB device has transitioned to the suspend state, the function sends USB_STS_SUSPEND as the return value.

5.4 USB_STS_RESUME

When the R_USB_GetEvent function is called after USB device in the suspend state resumes by the resume signal, the function sends USB_STS_RESUME as the return value.

5.5 USB_STS_DETACH

When the R_USB_GetEvent function is called after the USB device has been detached from the USB host, the function sends USB_STS_DETACH as the return value.

5.6 USB_STS_REQUEST

When the R_USB_GetEvent function is called after the USB device has received a USB request (Setup), the function sends USB_STS_REQUEST as the return value. Information is also set in the following usb_ctrl_t structure member.

Member	Description
setup	Received USB request information (8 bytes)

Note:

1. When a request has been received for support of the no-data control status stage, even if the R_USB_GetEvent function is called, USB_STS_REQUEST_COMPLETE is sent as the return value instead of USB_STS_REQUEST.
2. For more details on USB request information (8 bytes) stored in member (setup), refer to chapter 8.2, usb_setup_t structure.

5.7 USB_STS_REQUEST_COMPLETE

After the status stage of a control transfer is complete and transition to the idle stage has occurred, if the R_USB_GetEvent function is called, then USB_STS_REQUEST_COMPLETE will be the return value. In addition to this, the following member of the usb_ctrl_t structure also has information.

Member	Description
status	Sets either USB_ACK / USB_STALL

Note:

When a request has been received for support of the no-data control status stage, USB request information (8 bytes) is stored in the usb_ctrl_t structure member (setup). For more details on USB request information (8 bytes) stored in member (setup), refer to chapter 8.2, usb_setup_t structure.

5.8 USB_STS_READ_COMPLETE

When the R_USB_GetEvent function is called after a data read has been completed in the R_USB_Read function, USB_STS_READ_COMPLETE is sent as the return value. Information is also set in the following usb_ctrl_t structure member.

Member	Description
type	Device class type of completed data read (only set when using R_USB_Read function)
size	Size of read data
pipe	Pipe number of completed data read
status	Read completion error information

Note:

1. In the case of the R_USB_PipeRead function, the member (pipe) has the PIPE number (USB_PIPE1 to USB_PIPE9) for which data read is completed. In the case of the R_USB_Read function, USB_NULL is set to the member (pipe).
2. For details on device class type, refer to chapter 6, Device Class Types.
3. The member (status) has the read completion error information. The error information set to this member is as follows.

Member	Description
USB_SUCCESS	Data read successfully completed
USB_ERR_OVER	Received data size over
USB_ERR_SHORT	Received data size short
USB_ERR_NG	Data reception failed

(1). Even if the reception request size is less than MaxPacketSize × n, if MaxPacketSize × n bytes of data are received, then USB_ERR_OVER is set.

For example, if MaxPacketSize is 64 bytes, the specified reception request size is 510 bytes (less than MaxPacketSize × n), and the actual received data size is 512 bytes (MaxPacketSize × n), then USB_ERR_OVER is set.

(2). If the reception request size is less than MaxPacketSize × n and the actual received data size is less than this reception request size, then USB_ERR_SHORT is set.

For example, if MaxPacketSize is 64 bytes, the specified reception request size is 510 bytes, and the actual received data size is 509 bytes, then USB_ERR_SHORT is set.

(3). The read data size is set in the member size when the read completion error information is USB_SUCCESS or USB_ERR_SHORT.

5.9 USB_STS_WRITE_COMPLETE

When the R_USB_GetEvent function is called after a data write has been completed in the R_USB_Write function, USB_STS_WRITE_COMPLETE is sent as the return value. Information is also set in the following usb_ctrl_t structure member.

Member	Description
type	Device class type of completed data write (only set when using R_USB_Write function)
pipe	Pipe number of completed data write
status	Write completion error information

Note:

1. For R_USB_Write function: class type is set in the usb_ctrl_t structure member (type) and USB_NULL is set in the member (pipe).
2. In the case of R_USB_PipeWrite function, the member (pipe) has the PIPE number (USB_PIPE1 to USB_PIPE9) for which data write has been completed. In the case of the R_USB_Write function, USB_NULL is set to the member (pipe).
3. For details on device class type, refer to chapter 6, Device Class Types.
4. The member (status) has the write completion error information. The error information set to this member is as follows.

Status	Description
USB_SUCCESS	Data write successfully completed
USB_ERR_NG	Data transmission failed

5.10 USB_STS_NONE

When the R_USB_GetEvent function is called in the “no USB-related event” status, USB_STS_NONE is sent as the return value. Information is also set in the following usb_ctrl_t structure member.

Member	Description
status	USB device status

6. Device Class Types

The device class types assigned to the member(type) of the usb_ctrl_t and usb_info_t structures are as follows. Please specify the device class supported by your system.

Device class type	Description
USB_PCDC	Peripheral Communication Device Class
USB_PCDCC	Peripheral Communication Device Class (Control Class)
USB_PHID	Peripheral Human Interface Device Class
USB_PMSC	Peripheral Mass Storage Device Class
USB_PVND	Peripheral Vendor Class

Note:

1. Peripheral Communication Device Class: When transmitting data in a bulk transfer, specify USB_PCDC in the usb_ctrl_t structure member (type). When transmitting data in an interrupt transfer, specify USB_PCDCC in the usb_ctrl_t structure member (type).
2. For an application program, do not assign USB_HMSC, USB_PMSC, USB_HVND, and USB_PVND to the member (type) of the usb_ctrl_t structure.

7. Configuration

7.1 SmartConfigurator Configuration(r_usbf_basic_drv_sc_cfg.h)

Specify the following definitions using SmartConfigurator.

1. OS / OSLess select

Select OSLess / FreeRTOS.

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

2. LPM select

Select LPM use / not use.

```
#define       USE_LPM      0      // not LPM
#define       USE_LPM      1      // LPM
```

3. USB module select

Select module USB0 / USB1.

```
#define USB_CFG_USE_USBIP  USB_CFG_IP0  // USB0 select
#define USB_CFG_USE_USBIP  USB_CFG_IP1  // USB1 select
```

7.2 USB Peripheral Basic Configurations(r_usb_basic_config.h)

Perform settings for the definitions below.

1. Argument check setting

Specify whether to perform argument checking for all of the APIs listed in chapter 4, API Functions.

```
#define USB_CFG_PARAM_CHECKING    USB_CFG_ENABLE     // Checks arguments.
#define USB_CFG_PARAM_CHECKING    USB_CFG_DISABLE    // Does not check arguments.
```

2. Device class setting

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

```
#define USB_CFG_PCDC_USE   // Peripheral Communication Device Class
#define USB_CFG_PHID_USE   // Peripheral Human Interface Device Class
#define USB_CFG_PMSC_USE   // Peripheral Mass Storage Class
#define USB_CFG_PVNDR_USE  // Peripheral Vendor Class
```

3. DMA use setting

Specify whether to use the DMA.

```
#define USB_CFG_DMA  USB_CFG_ENABLE     // Uses DMA.
#define USB_CFG_DMA  USB_CFG_DISABLE    // Does not use DMA.
```

Note:

(1). If USB_CFG_ENABLE is set for the definition of USB_CFG_DMA, set the DMA Channel number for the definition in 4 below.

4. DMA Channel setting

If USB_CFG_ENABLE is set in 5 above, set the DMA Channel number to be used.

```
#define USB_CFG_USB0_DMA_TX      DMA Channel number  // Transmission setting for
                                                        USB0 module
#define USB_CFG_USB0_DMA_RX      DMA Channel number  // Transmission setting for
                                                        USB0 module
#define USB_CFG_USB1_DMA_TX      DMA Channel number  // Transmission setting for
                                                        USB1 module
#define USB_CFG_USB1_DMA_RX      DMA Channel number  // Transmission setting for
                                                        USB1 module
```

Note:

(1). Set one of the DMA channel numbers from USB_CFG_CH0 to USB_CFG_CH7. Do not set the same DMA Channel number.

(2). If DMA transfer is not used, set USB_CFG_NOUSE as the DMA Channel number.

5. CPU bus wait setting

Assign the value to be set for the BUSWAIT bit of the SYSCFG1 register in the USB module as the definition of USB_CFG_BUSWAIT.

```
#define USB_CFG_BUSWAIT          7     // Set to 7 wait cycles
```

Note:

(1). For the calculation of the value to be set for USB_CFG_BUSWAIT, refer to the chapter of the BUSWAIT bit of the SYSCFG1 register in the RZ/A2M hardware manual.

6. Interrupt Priority Level setting

Assign the interrupt priority level of the interrupt related to USB for USB_CFG_INTERRUPT_PRIORITY definition.

```
#define USB_CFG_INTERRUPT_PRIORITY     3     // 1(low) – 15(high)
```

7. USB module selection setting

Set the USB module number to be used for the definition of USB_CFG_USE_USBIP.

```
#define USB_CFG_USE_USBIP        USB_CFG_IP0        // Uses USB0 module
#define USB_CFG_USE_USBIP        USB_CFG_IP1        // Uses USB1 module
```

Note:

If the MCU being used only supports one USB module, then set USB_CFG_IP0 for the definition of USB_CFG_USE_USBIP.

8. Setting class request

Set whether the received class request is supported. If USB_CFG_ENABLE (supported) is set, then the USB driver will notify the reception of the class request to the application program. If USB_CFG_DISABLE (not supported) is set, then the USB driver will respond a STALL to the class request.

```
#define USB_CFG_CLASS_REQUEST    USB_CFG_ENABLE     // Supported
#define USB_CFG_CLASS_REQUEST    USB_CFG_DISABLE    // Not supported
```

Note:

a. Check the return value (USB_STS_REQUEST) of R_USB_GetEvent function when confirming whether USB driver receive the class request or not.

b. Even if USB_CFG_DISABLE is set, USB driver return the value "1" to GetMaxLun class request of Mass storage class.

9. Setting power saving function

   Set the power saving function to be enabled or disabled as the definition below. If USB_CFG_ENABLE is set as the definition below, then when there is a transition to suspend state or detach state, the USB driver will transition the MCU to power saving mode.

```
#define USB_CFG_LPW        USB_CFG_ENABLE     // Power saving function enabled.
#define USB_CFG_LPW        USB_CFG_DISABLE    // Power saving function disabled.
```

7.3 Other Definitions

In addition to the above, the following definitions 1 through 2 are also provided in r_usb_basic_config.h. Recommended values have been set for these definitions, so only change them when necessary.

1. DBLB bit setting

   Set or clear the DBLB bit in the pipe configuration register (PIPECFG) of the USB module using the following definition.

```
#define USB_CFG_DBLB       USB_CFG_DBLBON     // DBLB bit set.
#define USB_CFG_DBLB       USB_CFG_DBLBOFF    // DBLB bit cleared.
```

2. CNTMD bit setting

   Set or clear the CNTMD bit in the pipe configuration register (PIPECFG) of the USB module using the following definition.

```
#define USB_CFG_CNTMD      USB_CFG_CNTMDON    // CNTMD bit set.
#define USB_CFG_CNTMD      USB_CFG_CNTMDOFF   // CNTMD bit cleared.
```

Note:

(1). The setting of the DBLB and CNTMD bits above is performed for all the pipes being used. Therefore, in this configuration, it is not possible to perform the pipe-specific settings for these bits.
(2). For details on the pipe configuration register (PIPECFG), refer to the MCU hardware manual.
(3). Be sure to set SHTNAK bit.

8. Structures

This chapter describes the structures used in the application program.

8.1 usb_ctrl_t structure

The usb_ctrl_t structure is used for USB data transmission and other operations. The usb_ctrl_t structure can be used in all APIs listed in Table 4-1, excluding R_USB_GetVersion.

```
typdef struct usb_ctrl {
    uint8_t      module;      /* Note 1 */
    uint8_t      address;     /* Note 2 */
    uint8_t      pipe;        /* Note 3 */
    uint8_t      type;        /* Note 4 */
    uint16_t     status;      /* Note 5 */
    uint32_t     size;        /* Note 6 */
    usb_set_up   setup;       /* Note 7 */
} usb_ctrl_t;
```

Note:

1. Member (module) is used to specify the USB module number.
2. Member (address) is used to specify the USB device address.
3. Member (pipe) is used to specify the USB module pipe number. For example, specify the pipe number when using the R_USB_PipeRead function or R_USB_PipeWrite function.
4. Member (type) is used to specify the device class type.
5. The USB device state or the result of a USB request command is stored in the member (status). The USB driver sets in this member. Therefore, except when initializing the usb_crtl_t structure area or processing an ACK/STALL response to a vendor class request, the application program should not write into this member. For processing an ACK/STALL response to a vendor class request, see 9.1.5, Processing ACK/STALL Response to Class Request.
6. Member (size) is used to set the size of data that is read. The USB driver sets this member. Therefore, the application program should not write into this member.
7. Member (setup) is used to set the information about a class request.

8.2 usb_setup_t structure

The usb_setup_t structure is used when sending or receiving a USB class request. To obtain class request information from the USB Host (in USB Peripheral mode), refer to the members of the usb_setup_t structure.

```
typedef struct usb_setup {
    uint16_t     type;        /* Note 1 */
    uint16_t     value;       /* Note 2 */
    uint16_t     index;       /* Note 3 */
    uint16_t     length;      /* Note 4 */
} usb_setup_t;
```

Note:

1. In USB Peripheral mode, the value of the USBREQ register is set to the member (type).
2. In USB Peripheral mode, the value of the USBVAL register is set to the member (value).
3. In USB Peripheral mode, the value of the USBINDX register is set to the member (index).
4. In USB Peripheral mode, the value of the USBLENG register is set to the member (length).
5. For information on the USBREQ, USBVAL, USBINDX, and USBLENG registers, refer to the MCU user's manual.

8.3 usb_cfg_t structure

The usb_cfg_t structure is used to register essential information such as settings to indicate use of USB peripheral as the USB module and to specify USB speed. This structure can only be used for the R_USB_Open function listed in Table 4-1.

```
typdef struct usb_cfg {
    uint8_t             usb_mode;    /* Note 1 */
    uint8_t             usb_speed;   /* Note 2 */
    usb_descriptor_t    *p_usb_reg;  /* Note 3 */
} usb_cfg_t;
```

Note:

1. Specify to use USB peripheral mode as the USB module in member (usb_mode). To select USB peripheral, set USB_PERI in the member.
2. Specify the USB speed for USB module operations. Set "USB_HS" to select Hi-speed, "USB_FS" to select Full-speed.
3. Specify the usb_descriptor_t type pointer for the USB device in member (p_usb_reg). Refer to chapter 8.4, usb_descriptor_t structure for details on the usb_descriptor_t type. This member can only be set in USB peripheral mode.

8.4 usb_descriptor_t structure

The usb_descriptor_t structure stores descriptor information such as device descriptor and configuration descriptor. The descriptor information set in this structure is sent to the USB host as response data to a standard request during enumeration of the USB host. This structure is specified in the R_USB_Open function argument.

```
typdef struct usb_descriptor {
    uint8_t      *p_device;          /* Note 1 */
    uint8_t      *p_config_f;        /* Note 2 */
    uint8_t      *p_config_h;        /* Note 3 */
    uint8_t      *p_qualifier;       /* Note 4 */
    uint8_t      **p_string;         /* Note 5 */
    uint8_t      num_string;         /* Note 6 */
} usb_descriptor_t;
```

Note:

1. Specify the top address of the area that stores the device descriptor in the member (p_device).
2. Specify the top address of the area that stores the Full-speed configuration descriptor in the member (p_config_f). Even when using Hi-speed, make sure you specify the top address of the area that stores the Full-speed configuration descriptor in this member.
3. Specify the top address of the area that stores the Hi-speed configuration descriptor in the member (p_config_h). For Full-speed, specify USB_NULL to this member.
4. Specify the top address of the area that stores the qualifier descriptor in the member (p_qualifier). For Full-speed, specify USB_NULL to this member.
5. Specify the top address of the string descriptor table in the member (p_string). In the string descriptor table, specify the top address of the areas that store each string descriptor.

Ex. 1) Full-speed

```
usb_descriptor_t
usb_descriptor =
{
    smp_device,
    smp_config_f,
    USB_NULL,
    USB_NULL,
    smp_string,
    3,
};
```

Ex. 2) Hi-speed

```
usb_descriptor_t usb_descriptor =
{
    smp_device,
    smp_config_f,
    smp_config_h,
    smp_qualifier,
    smp_string,
    3,
};
```

6. Specify the number of the string descriptor which set in the string descriptor table to the member (num_string).

8.5 usb_pipe_t structure

The USB driver sets information about the USB pipe (PIPE1 to PIPE9) in the usb_pipe_t structure. Use the R_USB_GetPipeInfo function to reference the pipe information set in the structure.

```
typdef struct usb_pipe {
    uint8_t      ep;          /* Note 1 */
    uint8_t      type;        /* Note 2 */
    uint16_t     mxps;        /* Note 3 */
} usb_pipe_t;
```

Note:

1. The endpoint number is set in member (ep). The direction (IN/OUT) is set in the highest bit. When the highest bit is “1”, the direction is IN, when “0”, the direction is OUT.
2. The transfer type (bulk/interrupt) is set in member (type). For a Bulk transfer, "USB_BULK" is set, and for an Interrupt transfer, "USB_INT" is set.
3. The maximum packet size is set in member (mxps).

8.6 usb_info_t structure

The following information on the USB device is set for the usb_info_t structure by calling the R_USB_GetInformation function.

```
typedef struct usb_info {
    uint8_t     type;       /* Note 1 */
    uint8_t     speed;      /* Note 2 */
    uint8_t     status;     /* Note 3 */
    uint8_t     port;       /* Note 4 */
} usb_info_t;
```

Note:

1. In USB Peripheral mode, the supporting device class type is set for the member (type). For information on the device class types, see 6, Device Class Types. (In the case of PCDC, USB_PCDC is set in this member(type))
2. The USB speed (USB_HS/USB_FS/USB_LS) is set for the member (speed).
3. One of the following states of the USB device is set for the member (status).

Status	Description
USB_STS_DEFAULT	Default state
USB_STS_ADDRESS	Address state (USB Peripheral only)
USB_STS_CONFIGURED	Configured state
USB_STS_SUSPEND	Suspend state
USB_STS_DETACH	Detach state

4. The following information of the Battery Charging (BC) function of the device conected to the port is set to the member (port).

Port	Description
USB_SDP	Standard Downstream Port
USB_CDP	Charging Downstream Port
USB_DCP	Dedicated Charging Port (USB Peripheral only)

8.7 usb_compliance_t structure

This structure is used when running the USB compliance test. The structure specifies the following USB-related information:

```
typedef struct usb_compliance {
    usb_ct_status_t     status;     /* Note 1 */
    uint16_t            vid;        /* Note 2 */
    uint16_t            pid;        /* Note 3 */
} usb_compliance_t;
```

Note:

1. The member status can be set to the following values to indicate the status of the connected USB device:

Status	Description
USB_CT_ATTACH	USB device attach detected
USB_CT_DETACH	USB device detach detected
USB_CT_TPL	Attach detected of USB device listed in TPL
USB_CT_NOTTPL	Attach detected of USB device not listed in TPL
USB_CT_HUB	USB hub connection detected
USB_CT_OVRCUR	Overcurrent detected
USB_CT_NORES	No response to control read transfer
USB_CT_SETUP_ERR	Setup transaction error occurred

2. The member vid is set to a value indicating the vendor ID of the connected USB device.
3. The member pid is set to a value indicating the product ID of the connected USB device.

9. USB Class Requests

This chapter describes how to process USB class requests. As standard requests are processed by the USB driver, they do not need to be included in the application program.

9.1 USB Peripheral operations

9.1.1 USB request (Setup)

Confirm receipt of the USB request (Setup) sent by the USB host with the return value (USB_STS_REQUEST) of the R_USB_GetEvent function. The contents of the USB request (Setup: 8 bytes) are stored in the usb_ctrl_t structure member (setup) area. Refer to chapter 8.2, usb_setup_t structure for a description of the settings for member (setup).

Note:

The return value of the R_USB_GetEvent function when a request that supports the no-data control status stage is received is USB_STS_REQUEST_COMPLETE, not USB_STS_REQUEST.

9.1.2 USB request data

The R_USB_Read function is used to receive data in the data stage and the R_USB_Write function is used to send data to the USB host. The following describes the receive and send procedures.

1. Receive procedure
   (1). Set the USB_REQUEST in the usb_ctrl_t structure member (type).
   (2). In the R_USB_Read function, specify the pointer to area that stores data in the second argument, and the requested data size in the third argument.
   (3). Call the R_USB_Read function.
2. Send procedure
   (1). Set USB_REQUEST in the usb_ctrl_t structure member (type).
   (2). Store the data from the data stage in a buffer. In the R_USB_Write function, specify the top address of the buffer in the second argument, and the transfer data size in the third argument.
   (3). Call the R_USB_Write function.
   Note:

   Confirm receipt of the request data with the return value (USB_STS_WRITE_COMPLETE) of the R_USB_GetEvent function. You can also confirm whether the usb_ctrl_t structure member (type) has been set to USB_REQUEST.

9.1.3 USB request results

For each class, if USB_CFG_ENABLE is set as the definition of the class request setting (example: USB_CFG_PCDC_REQUEST) in the configuration file (example: r_usb_pcdc_config.h), then this USB driver will always respond with an ACK to a received class request.

Note:

For a vendor class request, the USB driver does not respond with an ACK or STALL. An application program must respond with an ACK or STALL to the vendor class request. For how to respond with an ACK or STALL, see 9.1.5, Processing ACK/STALL Response to Class Request.

9.1.4 Example USB request processing description

1. Request that supports control read data stage

```
void usr_application (void)
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_REQUEST: /* Receive USB request */
            /* ctrl.setup analysis processing*/
                     :
            /* data setup processing */
                     :
            ctrl.type = USB_REQUEST;
            R_USB_Write(&ctrl, g_buf, size); /* data (data stage) send request */
        break;
        case USB_STS_REQUEST_COMPLETE:
                     :
        break;
                     :
    }
}
```

2. Request that supports control write data stage

```
void usr_application (void )
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_REQUEST: /* Receive USB request */
            /* ctrl.setup analysis processing */
                     :
            ctrl.type = USB_REQUEST;
            R_USB_Read(&ctrl, g_buf, size); /* data (data stage) receive request */
        break;
        case USB_STS_REQUEST_COMPLETE:
                     :
        break;
                     :
    }
}
```

3. Request that supports no-data control status stage

```
void usr_application (void)
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_REQUEST: /* Receive USB request */
            /* ctrl.setup analysis processing*/
                     :
            ctrl.type   = USB_REQUEST;
            ctrl.status = USB_ACK;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
        break;
        case USB_STS_REQUEST_COMPLETE:
                     :
        break;
                     :
    }
}
```

4. Example of processing STALL response

```
void usr_application (void)
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_STS_REQUEST:
            /* ctrl.setup analysis processing */
                     :
            ctrl.type   = USB_REQUEST:
            ctrl.status = USB_STALL;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
        break;
        case USB_STS_REQUEST_COMPLETE:
            if( USB_REQUEST == ctrl.type )
            {
                     :
            }
        break;
                     :
    }
}
```

9.1.5 Processing ACK/STALL Response to Class Request

When it is necessary to respond with ACK or STALL to a class request, assign USB_REQUEST to the member(type) of the usb_ctrl_t structure, and either USB_ACK or USB_STALL to the member (status), and call the R_USB_Write function. Assign USB_NULL to both the second and third arguments of the R_USB_Write function. The completion of transmission of ACK/STALL can be checked by reading the USB_STS_REQUEST_COMPLETE return value of the R_USB_GetEvent function. At this time, check also that USB_REQUEST has been set for the member (type) of the usb_ctrl_t structure.

1. Example of processing STALL response

```
void usr_application (void )
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_STS_REQUEST:
            /* ctrl.setup analysis processing */
                     :
            ctrl.type   = USB_REQUEST:
            ctrl.status = USB_STALL;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
            break;
        case USB_STS_REQUEST_COMPLETE:
            if( USB_REQUEST == ctrl.type )
            {
                            :
            }
            break;
    }
}
```

2. Example of processing ACK response

```
void usr_application (void )
{
    usb_ctrl_t   ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                     :
        case USB_STS_REQUEST:
            /* ctrl.setup analysis processing */
                     :
            ctrl.type   = USB_REQUEST:
            ctrl.status = USB_ACK;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
            break;
        case USB_STS_REQUEST_COMPLETE:
            if( USB_REQUEST == ctrl.type )
            {
                            :
            }
            break;
    }
}
```

10. DMA Transfer

10.1 Basic Specification

The specifications of the DMA transfer sample program code included in USB-BASIC-F/W are listed below.

USB Pipe1 and Pipe2 can used DMA access.

Table10-1 shows DMA Setting Specifications.

Table10-1 DMA Setting Specifications

Setting	Description
FIFO port used	D0FIFO and D1FIFO port
Transfer mode	Single transfer mode
Chain transfer	Disabled
Address mode	Full address mode
Read skip	Disabled
Access bit width (MBW)	4-byte transfer: 32-bit width
USB transfer type	BULK transfer
Transfer end	Receive direction: BRDY interrupt Transmit direction: D0FIFO/D1FIFO interrupt, BEMP interrupt

10.2 Notes

10.2.1 Data Reception Buffer Size

The user needs to allocate the buffer area for the following size to store the receiving data.

(1). When USB_CFG_CNTMDON is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h

Allocate the area more than n times FIFO buffer size. For FIFO buffer size, refer to the chapter 11.4, Reference or Change of PIPEBUF Register

(2). When USB_CFG_CNTMDOFF is specified for USB_CFG_CNTMD definition in r_usb_basic_config.h.

Allocate the area n times the max packet size.

10.2.2 USB Pipe

USB pipe which is used by DMA transfer is only PIPE1 and PIPE2. This driver does not work properly when USB pipe except PIPE1 and PIPE2 is used for DMA transfer. When data transfer is performed by combining DMA transfer and CPU transfer, use PIPE1 or PIPE2 for DTM transfer and use PIPE3, PIPE4 or PIPE5 for CPU transfer.

11. Additional Notes

11.1 Vendor ID

Be sure to use the user's own Vendor ID for the one to be provided in the Device Descriptor.

11.2 Compliance Test

In order to run the USB Compliance Test it is necessary to display USB device–related information on a display device such as an LCD. When the USB_CFG_COMPLIANCE definition in the configuration file (r_usb_basic_config.h) is set to USB_CFG_ENABLE, the USB driver calls the function (usb_compliance_disp) indicated below. This function should be defined within the application program, and the function should contain processing for displaying USB device–related information, etc.

Function name	void usb_compliance_disp(usb_compliance_t *);
Argument	usb_compliance_t * Pointer to structure for storing USB information

Note:

1. The USB driver sets the USB device–related information in an area indicated by an argument, and the usb_compliance_disp function is called.
2. For information on the usb_compliance_t structure, refer to 8.7, usb_compliance_t structure.
3. For a program sample of the usb_compliance_disp function, see 13.1, usb_compliance_disp function.

11.3 Reference or Change of PIPEBUF Register

Recommended values are set to the BUFSIZE and BUFNMB bits of the PIPEBUF register. When refering or changing to these bits, refer or change the following variables in the USB driver.

Device Class	File Name	Variable Name
Peripheral Communication Device Class Peripheral Human I/F Device Class Peripheral Mass Storage Class	r_usb_peptable.c	g_usb_eptbl

12. Creating an Application Program

This chapter explains how to create an application program using the API functions described throughout this document. Please make sure you use the API functions described here when developing your application program.

12.1 Configuration

Set each configuration file (header file) in the r_config folder to meet the specifications and requirements of your system. Please refer to chapter 7, Configuration(r_usb_basic_config.h) about setting of the configuration file.

12.2 Descriptor Creation

For USB peripheral operations, your will need to create descriptors to meet your system specifications. Register the created descriptors in the usb_descriptor_t function members.

12.3 Application Program Creation

12.3.1 Include

Make sure you include the following files in your application program.

1. r_usb_basic_if.h (Inclusion is obligatory.)
2. r_usb_xxxxx_if.h (I/F file provided for the USB device class to be used)
3. Include any other driver-related header files that are used within the application program.

12.3.2 Initialization

1. MCU pin settings

USB input/output pin settings are necessary to use the USB controller. The following is a list of USB pins that need to be set. Set the following pins as necessary.

Table12-1 USB I/O Pin Settings for USB Peripheral Operation

Pin Name	I/O	Function
USB_VBUS	input	VBUS pin for USB communication

Note:

(1). Please refer to the corresponding MCU user's manual for the pin settings in ports used for your application program.

2. USB-related initialization

Call the R_USB_Open function to initialize the USB module (hardware) and USB driver software used for your application program.

12.3.3 Descriptor Creation

For USB peripheral operations please create descriptors to meet your system specifications. Refer to chapter 3.5, Descriptor for more details about descriptors.

12.3.4 Main routine

Please describe the main routine in the main loop format. Make sure you call the R_USB_GetEvent function in the main loop. The USB-related completed events are obtained from the return value of the R_USB_GetEvent function. Also make sure your application program has a routine for each return value. The routine is triggered by the corresponding return value

12.3.5 Application program description example (CPU transfer)

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

void usb_peri_application( void )
{
    usb_ctrl_t   ctrl;
    usb_cfg_t    cfg;

    /* MCU pin setting */
    usb_pin_setting();

    /* Initialization processing */
    ctrl.module   = USB_IP1;           /* Specify the selected USB module */
    cfg.usb_mode  = USB_PERI;          /* Specify USB peri */
    cfg.usb_speed = USB_HS;            /* Specify the USB speed */
    cfg.p_usb_reg = &smp_descriptor;   /* Specify the top address of the descriptor table */
    R_USB_Open( &ctrl, &cfg );

    /* main routine */
    while(1)
    {
        switch( R_USB_GetEvent( &ctrl ) )
        {
             case USB_STS_CONFIGURED:
             case USB_STS_WRITE_COMPLETE:
                 ctrl.type = USB_PCDC;
                 R_USB_Read( &ctrl, g_buf, 64 );
                 break;
             case USB_STS_READ_COMPLETE:
                 ctrl.type = USB_PCDC;
                 R_USB_Write( &ctrl, g_buf, ctrl.size );
                 break;
             default:
                 break;
        }
    }
}
```

12.3.6 Application program description example (DMA transfer)

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

void usb_peri_application( void )
{
    usb_ctrl_t    ctrl;
    usb_cfg_t     cfg;

    /* MCU pin setting */
    usb_pin_setting();

    /* Initialization processing */
    ctrl.module   = USB_IP0;            /* Specify the selected USB module */
    cfg.usb_mode  = USB_PERI;           /* Specify USB peri */
    cfg.usb_speed = USB_HS;             /* Specify the USB speed */
    cfg.p_usb_reg = &smp_descriptor;    /* Specify the top address of the descriptor table */
    R_USB_Open( &ctrl, &cfg );

    /* main routine */
    while(1)
    {
        switch( R_USB_GetEvent( &ctrl ) )
        {
             case USB_STS_CONFIGURED:
             case USB_STS_WRITE_COMPLETE:
                 ctrl.type = USB_PCDC;
                 R_USB_Read( &ctrl, g_buf, 64 );
                 break;
             case USB_STS_READ_COMPLETE:
                 ctrl.type = USB_PCDC;
                 R_USB_Write( &ctrl, g_buf, ctrl.size );
                 break;
             default:
                 break;
        }
    }
}
```

13. Program Sample

13.1 usb_compliance_disp function

```
void usb_compliance_disp (usb_compliance_t *p_info)
{
    uint8_t     disp_data[32];

    disp_data = (usb_comp_disp_t*)param;

    switch(p_info->status)
    {
        case USB_CT_ATTACH:                 /* Device Attach Detection */
            display("ATTACH ");
        break;

        case USB_CT_DETACH:                 /* Device Detach Detection */
            display("DETTACH");
        break;

        case USB_CT_TPL:                    /* TPL device connect */
            sprintf(disp_data,"TPL PID:%04x VID:%04x",p_info->pid, p_info->vid);
            display(disp_data);
        break;

        case USB_CT_NOTTPL:                 /* Not TPL device connect */
            sprintf(disp_data,"NOTPL PID:%04x VID:%04x",p_info->pid, p_info->vid);
            display(disp_data);
        break;

        case USB_CT_HUB:                    /* USB Hub connect */
            display("Hub");
        break;

        case USB_CT_NOTRESP:                /* Response Time out for Control Read Transfer */
            display("Not response");
        break;

        default:
        break;
    }
}
```

Note:

The display function in the above function displays character strings on a display device. It must be provided by the customer.

14. Reference Documents

User's Manual: Hardware

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

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

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

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

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

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

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

Technical Update/Technical News

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

User's Manual: Development Tools

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

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep.30.19	-	First edition issued.
1.10	Dec.17.19		Table 2.1 Operation Confirmation Conditions(1/2) Remove compiler option "-mthumb-interwork" Support both FreeRTOS / OSLess
1.20	June.30.20	P42	Support LPM

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Corporate Headquarters

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

Contact information

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