RZ/T1 Group

R01AN2632EJ0130
Rev.1.30
Dec 07, 2017

USB Peripheral Mass Storage Class Driver (PMSC)

Introduction

This application note describes the USB peripheral mass storage class driver. This module operates in combination with the USB Peripheral Basic firmware (USB-BASIC-FW). After a while calls this sample software PMSC.

The sample program of this application note is created based on "RZ/T1 group Initial Settings Rev.1.30". Please refer to "RZ/T1 group Initial Settings application note (R01AN2554EJ0130)" about operating environment.

Target Device

RZ/T1 Group

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

Related Documents

1. USB Revision 2.0 Specification
2. USB Mass Storage Class Specification Overview Revision 1.1
3. USB Mass Storage Class Bulk-Only Transport Revision 1.0, “BOT” protocol
   http://www.usb.org/developers/docs/
4. RZ/T1 Group User’s Manual: Hardware (Document No.R01UH0483EJ0130)
5. RZ/T1 Group Initial Settings (Document No.R01AN2554EJ0130)
6. USB Peripheral Basic Firmware (Document No.R01AN2630EJ0130)

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

USB Devices Page
http://www.renesas.com/prod/usb/

Content

1. Overview

The PMSC comprises the BOT protocol in USB Mass Storage Class. When combined with USB-BASIC-FW, it enables communication with a USB host as a BOT-compatible storage device.

This module supports the following functions.

- Response to mass storage device class requests from a USB host
- Response to storage commands which are encapsulated in the BOT protocol

Limitations

HMSC is subject to the following limitations.

The structures contain members of different types. (Depending on the compiler, this may cause address misalignment of structure members.)

Terms and Abbreviations

APL	:	Application program
BOT	:	Mass storage class Bulk Only Transport.
CBW	:	Command Block Wrapper
CSW	:	Command Status Wrapper
PCD	:	Peripheral control driver of USB-BASIC-FW
PMSC	:	Peripheral mass storage USB class driver
USB-BASIC-FW	:	USB Peripheral Basic firmware for Renesas USB device

2. Software Configuration

Figure 2-1 shows the configuration of PMSC, and Table 2-1 lists the modules.

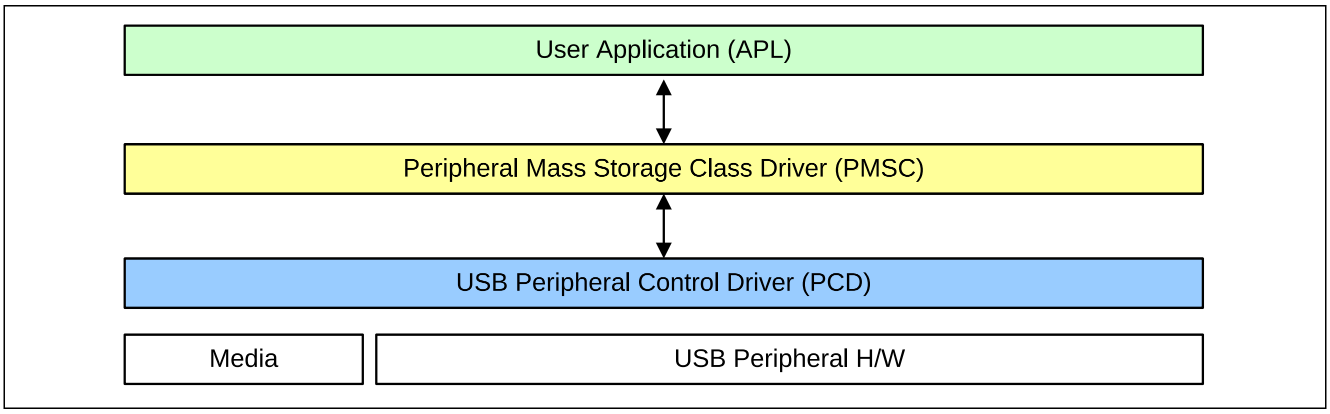

User Application (APL)
Peripheral Mass Storage Class Driver (PMSC)
USB Peripheral Control Driver (PCD)
Media
USB Peripheral H/W

Figure 2-1 Software Configuration Diagram

Table 2.1 Module Function Overview

Module	Description
APL	User application program (Please prepare for your system)
PMSC	Peripheral Mass Storage Class Driver ・ respond class request ・ control BOT protocol ・ receive and analyze CBW ・ processes storage commands ・ create and send CSW ・ accesses the media
PCD	USB Peripheral H/W Control driver (USB-BASIC-FW)

3. Peripheral Mass Storage Class Driver (PMSC)

3.1 Basic Functions

The functions of PMSC are as follows:

1. Respond to mass storage class requests from USB host.
2. Respond to USB host storage commands which are encapsulated in the BOT protocol.

3.2 Class Request

Table 3-1 lists the class requests supported by the PMSC.

Table 3.1 Supported MSC Class Requests

Request	bRequest	Description	Supported
Mass Storage Reset	0xFF	Resets the connection interface to the mass storage device.	Y
Get Max Lun	0xFE	Reports the logical numbers supported by the device.	Y

Y：Implemented　N：Not implemented(Stall response)

3.3 BOT Protocol Overview

The BOT is a transfer protocol that, encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out).

The storage commands and the response status are embedded in the CBW and the CSW.

Figure 3-1 shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

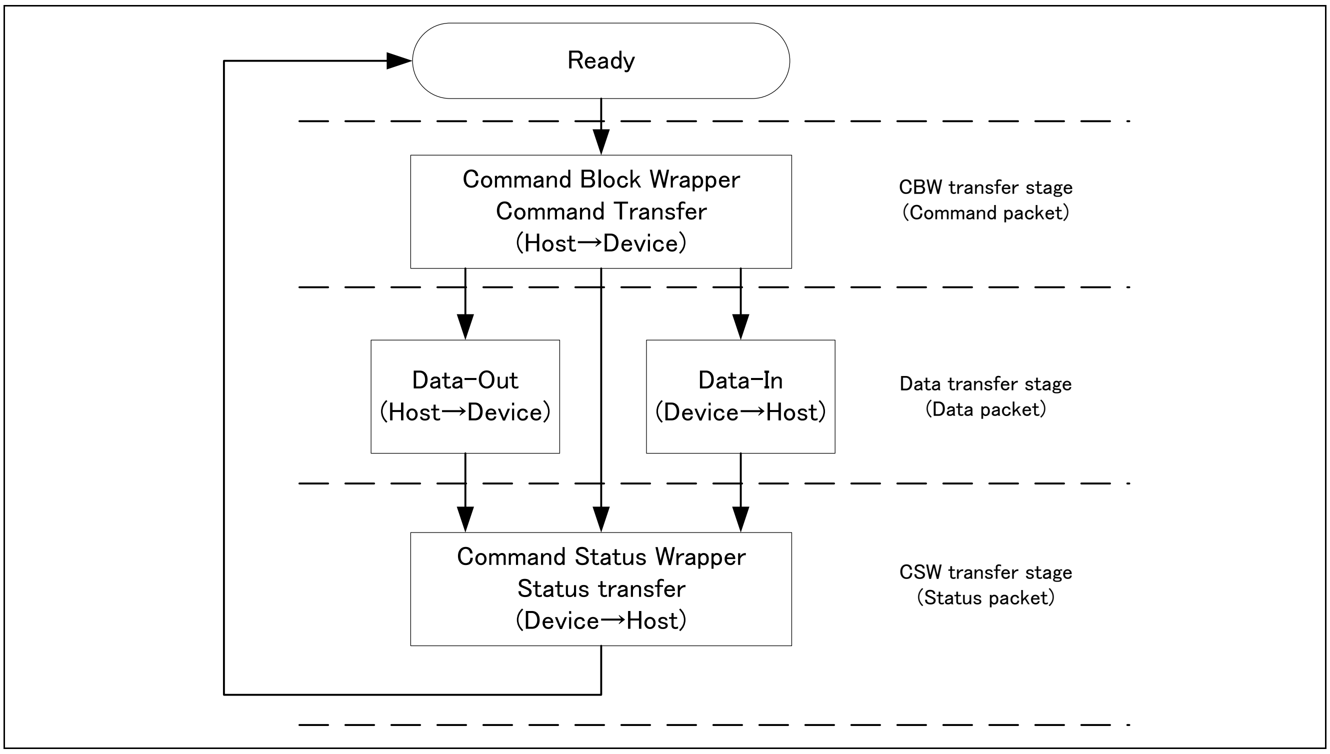


Figure 3-1 BOT protocol Overview

3.3.1 CBW processing

When PMSC receives a CBW from the host, it first verifies the validity of the CBW. If the CBW is valid, PMSC analysis the storage command contained in the CBW (CBWCB). PMSC finally performs processing based on the analysis (command validity, data transfer direction and size).

When the transfer data size exceeds USB_ATAPI_BLOCK_UNIT, the data is divided into smaller units and transferred.

Data transmission commands except READ10 is created from the response data table which prepared by PMSC.

The response data table follows storage command set.

Table 3-2 lists the storage commands supported by the PMSC.

Table 3.2 Supported Storage Commands

Command	Code	Description	Type	Supported
TEST_UNIT_READY	0x00	Checks the state of the peripheral device.	No Data	Y
REQUEST_SENSE	0x03	Gets the state of the peripheral device.	IN	Y
FORMAT_UNIT	0x04	Formats the logical unit.	OUT	N
INQUIRY	0x12	Gets the parameter information of the logical unit.	IN	Y
MODE_SELECT6	0x15	Specifies parameters.	OUT	N
MODE_SENSE6	0x1A	Gets the parameters of the logical unit.	IN	N
START_STOP_UNIT	0x1B	Enables/disabled logical unit access.	No Data	N
PREVENT_ALLOW	0x1E	Enables/disabled media removal.	No Data	Y
READ_FORMAT_CAPACITY	0x23	Gets the format table capacity.	IN	Y
READ_CAPACITY	0x25	Gets the capacity information of the logical unit.	IN	Y
READ10	0x28	Reads data.	IN	Y
WRITE10	0x2A	Writes data.	OUT	Y
SEEK	0x2B	Moves to a logical block address.	No Data	N
WRITE_AND_VERIFY	0x2E	Writes data with verification.	OUT	N
VERIFY10	0x2F	Verifies data.	No Data	N
MODE_SELECT10	0x55	Specifies parameters.	OUT	Y
MODE_SENSE10	0x5A	Gets the parameters of the logical unit.	IN	Y

Y：Implemented　N：Not implemented(Stall response)

3.3.2 Sequence of storage commands for no data transmit/receive

(a). CBW transfer stage

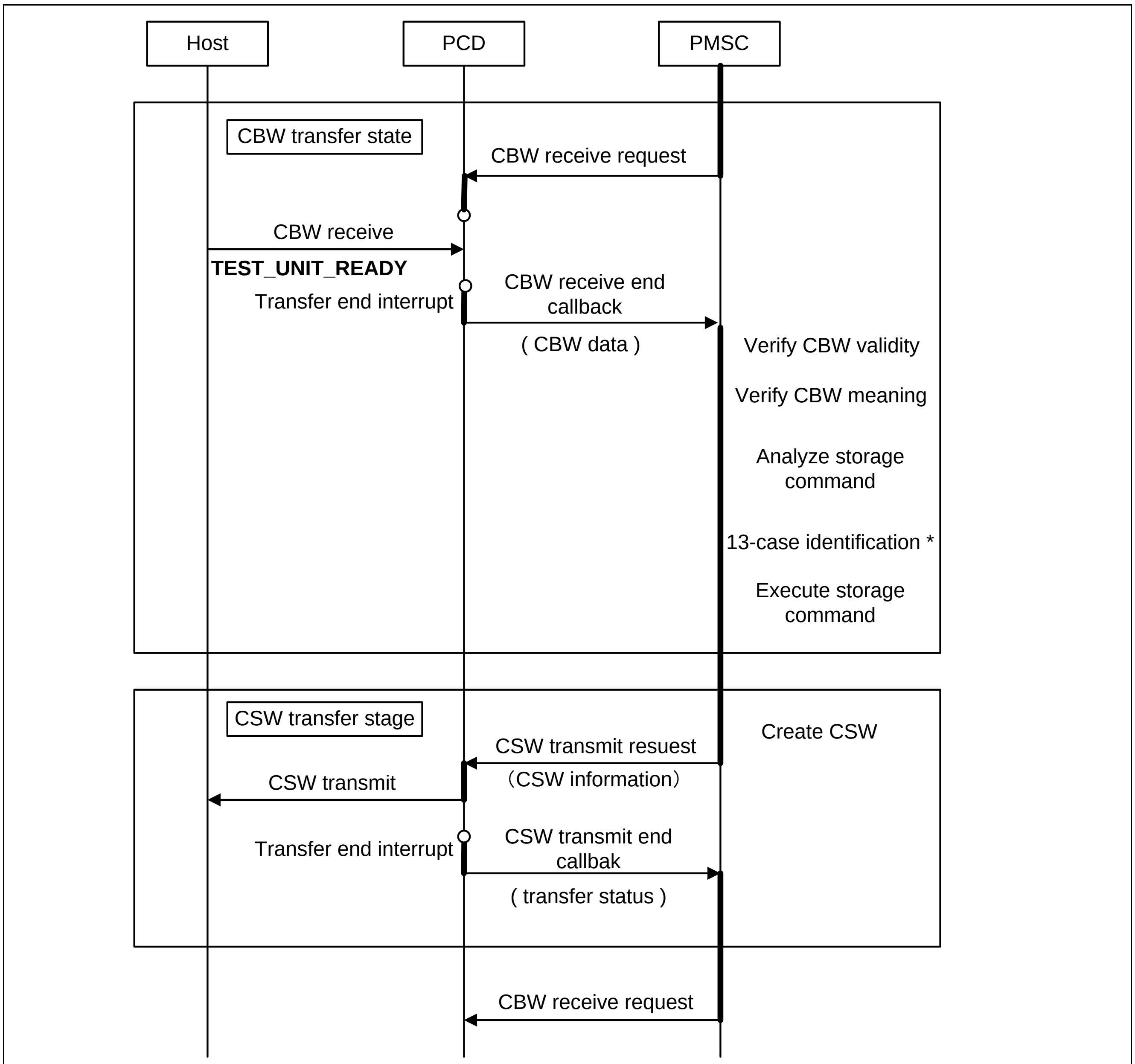
PMSC issues a CBW receive request to PCD. When PCD receives the CBW, it executes a callback function which starts the CBW transfer stage. PMSC verifies the validity of the CBW and analyzes the CBWCB.

PMSC confirms that the command is no data, compares the storage command analysis results and the information in CBW, execute the storage command.

(b). CSW transfer stage

PMSC creates a CSW based on the execution result and transmits it to the host via PCD.

Figure 3-2 shows the sequence.



* Error checking to classify 13 types of data transfer state between the host and peripherals.
Please refer to the 6.7 The Thirteen Cases of BOT specification details.

Figure 3-2 Sequence of storage commands for no data Transmit/Receive

3.3.3 Sequence of storage commands for transmit (IN) data

(a). CBW transfer stage

The same as 3.3.2 (a).

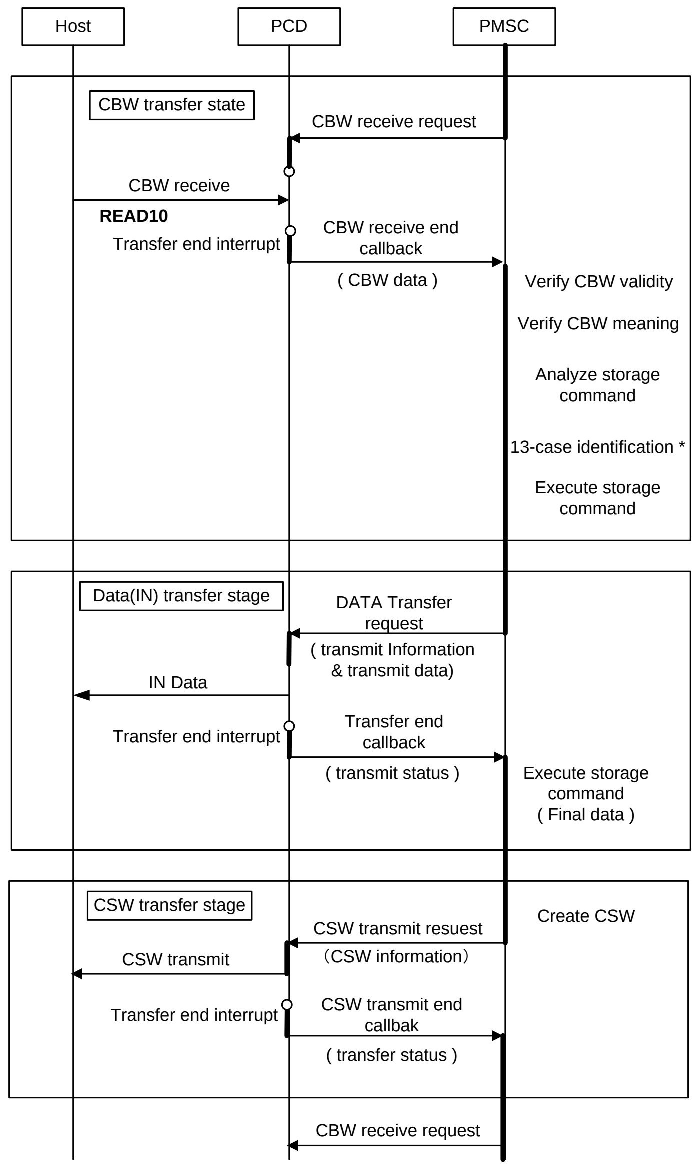
(b). Data IN transfer stage

PMSC notifies PCD of the data storage area and data size based on the execution result, and data communication with the USB host. PMSC the transmission completion is notified by the PCD, to verify that the transmission of the requested size is complete. If not completed, the DATA transmission request again to continue the DATA transfer stage. If completed, shifts to the CSW transfer stage.

(c). CSW transfer stage

The same as 3.3.2 (b).

Figure 3-3 shows the sequence.



* Error checking to classify 13 types of data transfer state between the host and peripherals.
Please refer to the 6.7 The Thirteen Cases of BOT specification details.

Figure 3-3 Sequence of Storage Commands for Transmit (IN) Data

3.3.4 Sequence of storage commands for receive (OUT) data

(a). CBW transfer stage

The same as 3.3.2 (a).

(b). Data OUT transfer stage

PMSC notifies PCD of the data storage area and data size based on the execution result, and data communication with the USB host. PMSC the reception completion is notified by the PCD, to verify that the reception of the requested size is complete. If not completed, the DATA reception request again to continue the DATA transfer stage. If completed, shifts to the CSW transfer stage.

(c). CSW transfer stage

The same as 3.3.2 (b).

Figure 3-4 shows the sequence.

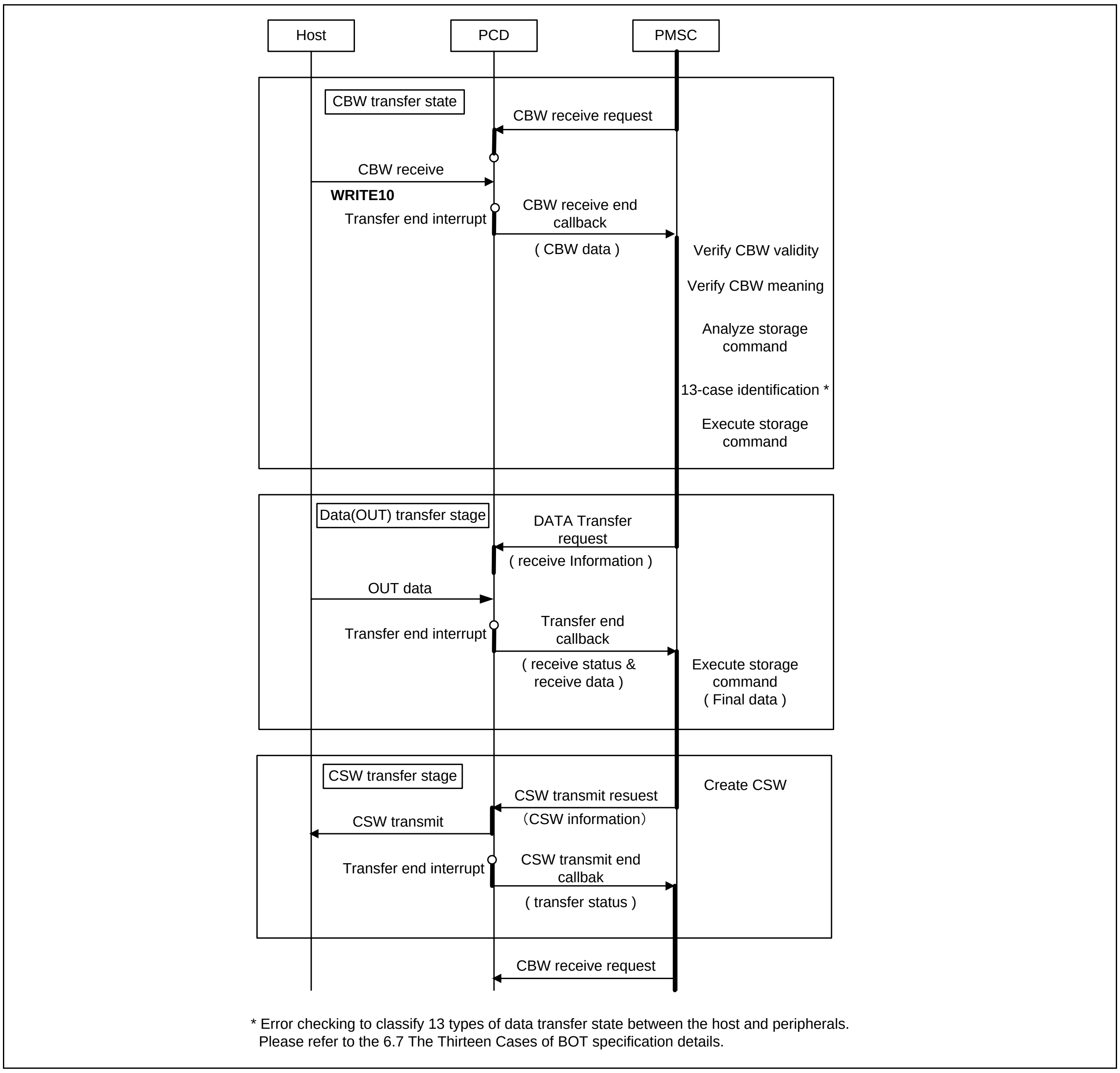


Figure 3-4 Sequence of Storage Command for Receive (OUT) Data

3.3.5 Sequence of class request

(a). Setup Stage

When PCD receives the SETUP, the process moves to the SETUP stage, to notify the reception in PMSC.

PMSC create a response data in according to the SETUP.

(b). Data Stage

PMSC executes the control transfer data stage and notifies PCD of data stage end by means of a callback function.

(c). Status Stage

PCD executes the status stage and ends the control transfer.

Figure 3-5 shows the sequence.

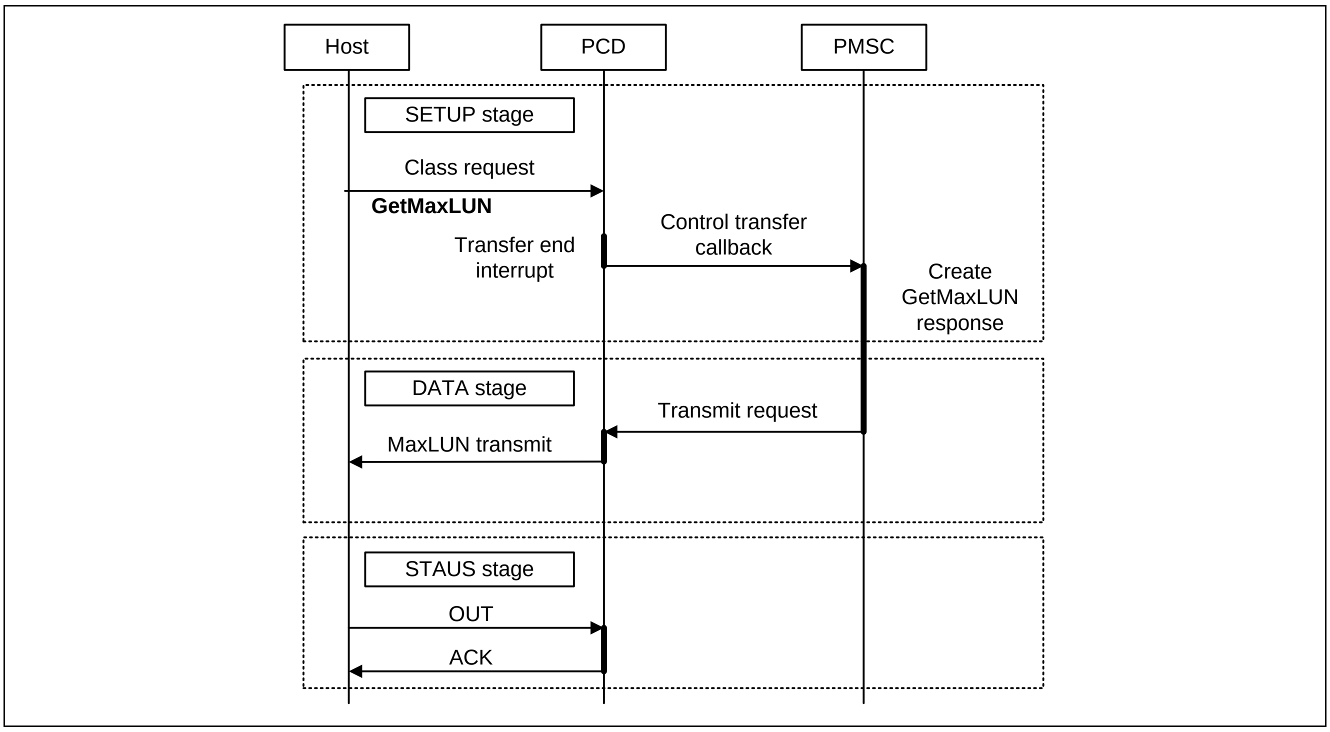


Figure 3-5 Sequence for Class Request

3.4 API

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

Please modify r_usb_pmsc_config.h when User sets the module configuration option.

Table 3-3 shows the option name and the setting value.

Table 3.3 Configuration options of PMSC

Define name	Default value	Description
USB_PMSC_USE_PIPE_IN	USB_PIPE1	Pipe number of IN transfer
USB_PMSC_USE_PIPE_OUT	USB_PIPE2	Pipe number of OUT transfer
USB_ATAPI_BLOCK_UNIT	0x200ul	ATAPI block size (byte unit)
USB_RAM_PP	0	Definition of RAM disk type
USB_SDRAM_PP	1	Definition of RAM disk type
USB_MEDIA_TYPE_PP	USB_SDRAM_PP	Setting of Media type
RAMDISK_MEDIA_SIZE	(64ul * 1024ul * 1024ul)	Size of RAM disk type (byte unit)
RAMDISK_SECT_SIZE	0x200ul	Sector size of RAM disk (byte unit)
RAMDISK_TOTALSECT	(RAMDISK_MEDIASIZE / RAMDISK_SECTSIZE)	number of RAM disk sector
MEDIA_ADDRESS	0x68000000	Header address of Media

Table 3-4 shows list API functions.

Table 3.4 List of API Functions

Function Name	Description
R_usb_pmsc_Open	Open PMSC
R_usb_pmsc_SetInterface	Processing of PMSC SET_INTERFACE
R_usb_pmsc_CtrlTrans	Processing of PMSC control transfer
R_usb_pmsc_poll	Processing of PMSC polling

3.4.1 R_usb_pmsc_Open

Open PMSC

Format

void R_usb_pmsc_Open(void)

Argument

—

Return Value

—

Description

This function is registered as a callback function to the member (devconfig) of USB_PCDREG_t structure.

This function sets the CBW reception setting.

Note

—

Example

```
void usb_pmsc_task_start( void )
{
 USB_PCDREG_t   driver;

 driver.devconfig  = &R_usb_pmsc_Open;
 R_usb_pstd_DriverRegistration(&driver);
}
```

3.4.2 R_usb_pmsc_Registration

Processing of PMSC SET_INTERFACE

Format

void	R_usb_pmsc_SetInterface(uin16_t data1)

Arguments

data1	Alternate number

Return Values

—

Description

This function is registered as a callback function to the member(interface) of USB_PCDREG_t structure.

This function sets the CBW reception setting.

Notes

—

Example

```
void usb_pmsc_task_start( void )
{
 USB_PCDREG_t   driver;

 driver.interface  = &R_usb_pmsc_SetInterface;
 R_usb_pstd_DriverRegistration(&driver);
}
```

3.4.3 R_usb_pmsc_CtrlTrans

Processing for MSC control transfer

Format

void	R_usb_pmsc_CtrlTrans (USB_REQUEST_t *preq, uint16_t ctsq)

Argument

*preq	Pointer to a class request message	
ctsq	Control transfer stage information	
	USB_CS_IDST	Idle or setup stage
	USB_CS_RDDS	Control read data stage
	USB_CS_WRDS	Control write data stage
	USB_CS_WRND	Control write no data status stage
	USB_CS_RDSS	Control read status stage
	USB_CS_WRSS	Control write status stage
	USB_CS_SQER	Sequence error

Return Value

—

Description

Register this API to the member “*ctrltrans*” in USB_PCDREG_t structure as the call-back function.

When the request type is a MSC class request, this function calls the processing that corresponds to the control transmit stage.

Note

—

Example

```
void usb_pmsc_task_start( void )
{
 USB_PCDREG_t   driver;

 driver.ctrltrans  = &R_usb_pmsc_CtrlTrans;
 R_usb_pstd_DriverRegistration(&driver);
}
```

3.4.4 R_usb_pmsc_poll

Processing of PMSC polling

Format

void R_usb_pmsc_poll(void)

Argument

—

Return Value

—

Description

Call this function in the main loop.

It is determined whether or not the transfer is complete, the case of the transfer is complete, and proceed with the BOT protocol processing sequence.

Note

—

Example

```
void usb_apl(void)
{
    while( 1 )
    {
        R_usb_pstd_poll();
        R_usb_pmsc_poll();
    }
}
```

4. Sample Application

This section describes the initial settings necessary for using the PMSC and USB-BASIC-F/W in combination as a USB driver and presents an example of data transfer by means of processing by the main routine and the use of API functions.

4.1 Operating Environment

Figure 4-1 shows an example operating environment for the PMSC.

Table 4.1 shows the OS environment in which the operation was confirmed.

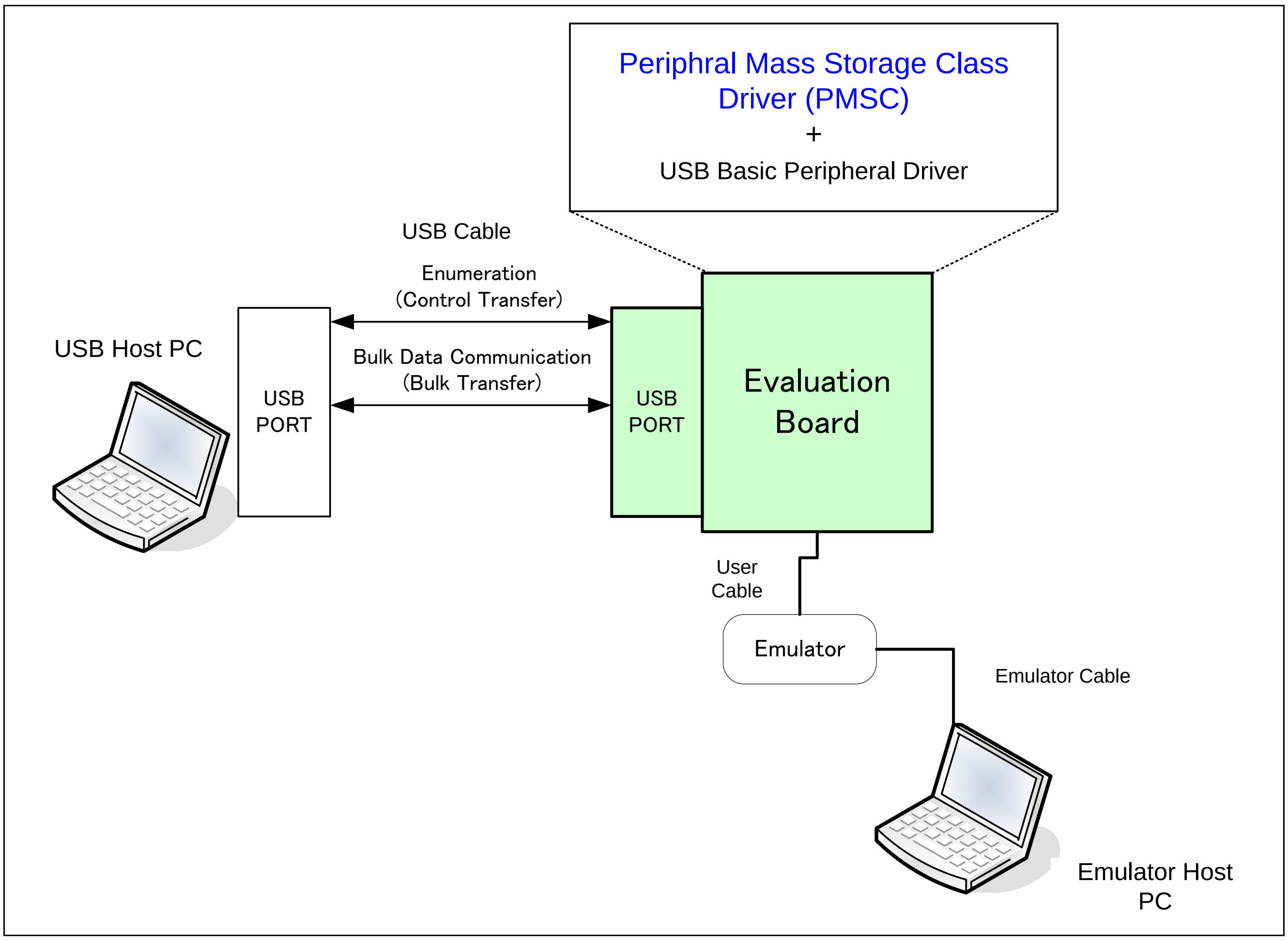


Figure 4-1 Example Operating Environment

Table 4.1 Operation Confirmed OS

OS Name	Remarks
Windows 7 32bit	—
Windows 7 64bit	—
Windows 8.1 32bit	—
Windows 8.1 64bit	—
Windows 10 32bit	—
Windows 10 64bit	—

4.2 Specifications

The sample application comprises two parts: initial settings and main loop.

The PMSC to process file write and file read to the storage area or the like to the request from the USB host. Therefore, the sample application performs no processing on data transferred from the host and only periodically call the USB driver.

Figure 4 2 shows a process flowchart of the sample application.

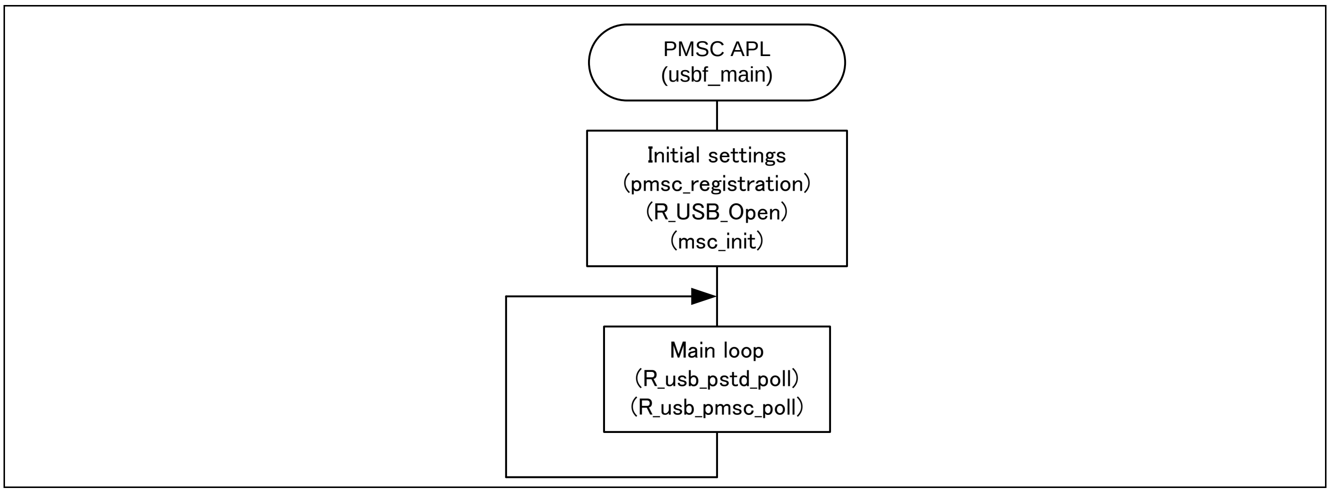
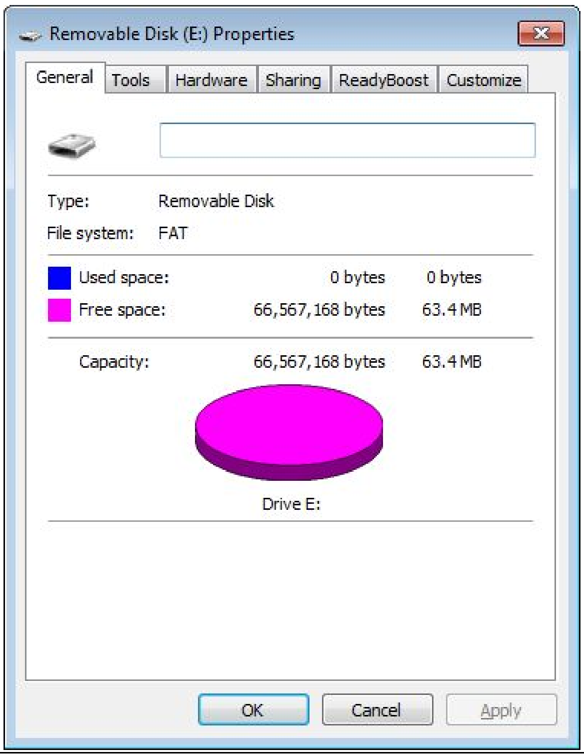


Figure 4-2 Flowchart

Sample application will be recognized as a removable disk when connected with the USB host. It is possible to perform the data transfer, such as file reading and writing.

Figure 4-3 shows the operating screen of a PC connection.an example operating environment for the PMSC.


Removable Disk (E:) Properties

General | Tools | Hardware | Sharing | ReadyBoost | Customize

Type:	Removable Disk	
File system:	FAT	
🟦 Used space:	0 bytes	0 bytes
🟪 Free space:	66,567,168 bytes	63.4 MB
Capacity:	66,567,168 bytes	63.4 MB

Figure 4-3 Operating screen

4.3 Initial settings

Sample settings are shown below.

```
void usbf_main(void)
{
    /* Initial setting of USB driver (Refer to “4.3.1”) */
    pmsc_registration();

    /* Startup USB module (Refer to “4.3.2”) */
    R_USB_Open();

    /* Initial setting of Application (Refer to “4.3.3”) */
    msc_init();

    /* main loop */
    while(1)
    {
        R_usb_pstd_poll();
        R_usb_pmsc_poll();
    }
}
```

4.3.1 Initial setting of USB driver

After specifying the necessary information in the members of the class driver registration structure (USB_PCDREG_t), call R_usb_pstd_DriverRegistration() to register the class driver information for the USB-BASIC-F/W.

Pipe information table and descriptor information is described in r_usb_pmsc_descriptor.c.

Create each descriptor based on USB specification.

A sample of information specified in the structure declared by USB_PCDREG_t is shown below.

```
void pmsc_registration(void)
{
 USB_PCDREG_t driver;     /* Structure for the class driver registration */

 /* Pipe information table setting */
 driver.pipetbl           = &usb_gpmsc_EpTbl[0];
 /* Set the Device Descriptor table */
 driver.devicetbl   = (uint8_t*)&usb_gpmsc_DeviceDescriptor;
 /* Set the Qualifier Descriptor table */
 driver.qualitbl    = (uint8_t*)&usb_gpmsc_QualifierDescriptor;
 /* Set the Configuration Descriptor table */
 driver.configtbl   = (uint8_t**)&usb_gpmsc_ConPtr;           // Note1
 /* Set the Other Configuration Descriptor */
 driver.othertbl    = (uint8_t**)&usb_gpmsc_ConPtrOther;      // Note1
 /* Set the String Descriptor */
 driver.stringtbl   = (uint8_t**)&usb_gpmsc_StrPtr;           // Note1
 /* Set the function which is called when changing to the default state */
 driver.devdefault  = &msc_default;
 /* Set the function which is called when completing the enumeration */
 driver.devconfig   = &msc_configured;
 /* Set the function which is called when disconnecting USB device */
 driver.devdetach   = &msc_detach;
 /* Set the function which is called when changing the suspend state */
 driver.devsuspend  = &msc_suspended;

 /* Set the function which is called when resuming from the suspend state */
 driver.devresume   = &msc_resume;
 /* Set the function which is called when changing the interface */
```

```
     driver.interface   = &R_usb_pmsc_SetInterface;
     /* Set the function which is called when processing the control transfer
     other than the standard request */
     driver.ctrltrans   = &R_usb_pmsc_CtrlTrans;

     /* Register the class driver information to PCD */
     R_usb_pstd_DriverRegistration(&driver);
   }
```

[Note]

1. Set the start address of array which is set the descriptor start address in this member.

```
[Example]
uint8_t *usb_gpmsc_StrPtr[] =
{
    usb_gpmsc_StringDescriptor0,
    usb_gpmsc_StringDescriptor1,
    usb_gpmsc_StringDescriptor2,
}
```

4.3.2 Startup USB module

Call the R_USB_Open() (API function of USB-BASIC-FW), set the USB module according to the initial setting sequence of the hardware manual, the USB interrupt handler registration and USB interrupt enable setting.

4.3.3 Initial setting of application

The sample application uses the SDRAM area in the media area of the removable disk.

It is implemented by assigning a global variable(g_ramdisk_mem[RAMDISK_MEDIASIZE]) in the file(r_ram_disk_format_data.c) to SDRAM area. If you want to change the media area, the memory arrangement in accordance with the operating environment, please change the defined values MEDIA_ADDRESS (see Table 3-3).

The SDRAM area cleared to zero at software startup. Then it is FAT16 file system formatted by writing a global variable(ram_disk_boot_sector[RAMDISK_SECTSIZE]) in file(r_ram_disk_format_data.c) to the top of the SDRAM area.

Appendix A. Changes of initial setting

USB-BASIC-F/W has been changed to "RZ/T1 group initial setting Rev.1.30".
Sample program supports IAR embedded workbench for ARM (EWARM) ,DS-5 and e^2 studio. However, Since it is not possible to use the SDRAM in RAM boot and is not supported.
This chapter describes the changes.

Folders and files

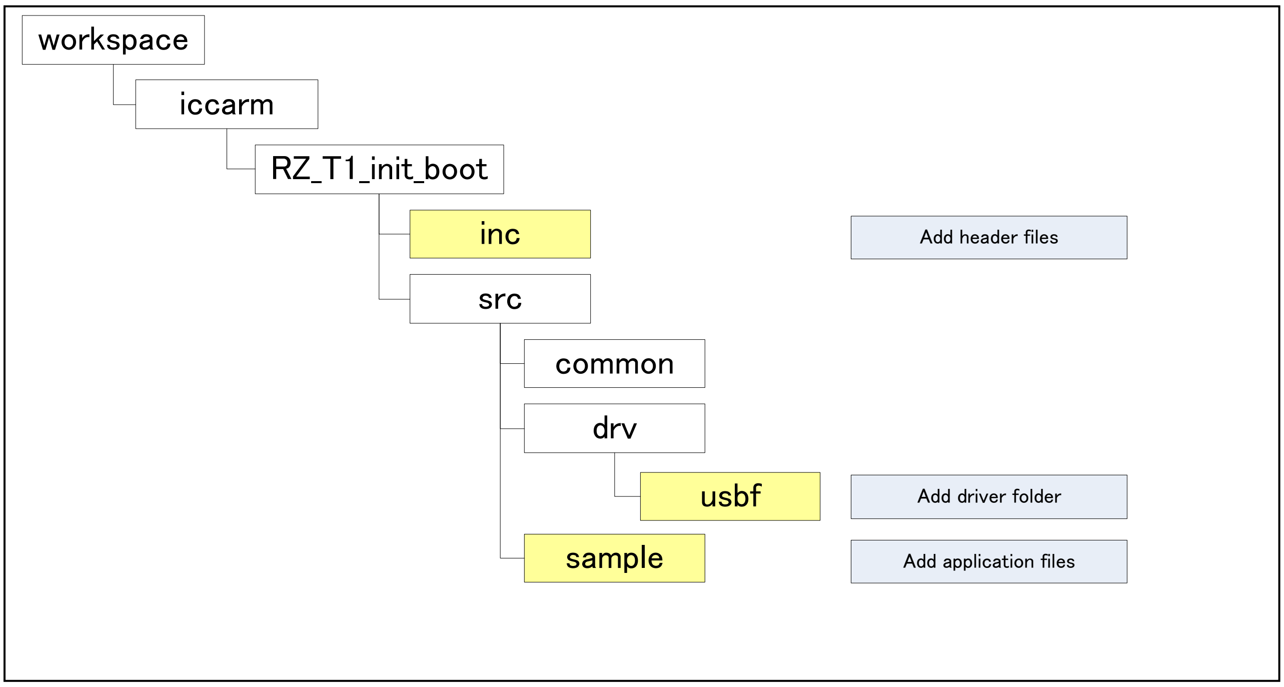
In the "RZ/T1 group initial setting Rev.1.30", different folder structure by the development environment and the boot method. Changes to each folder of all of the development environment and the boot method it is shown below.

・Add the following files in the “inc” folder.
r_usb_basic_config.h
r_usb_basic_if.h
r_usb_cdefusbip.h
r_usb_pmsc_config.h
r_usb_pmsc_if.h

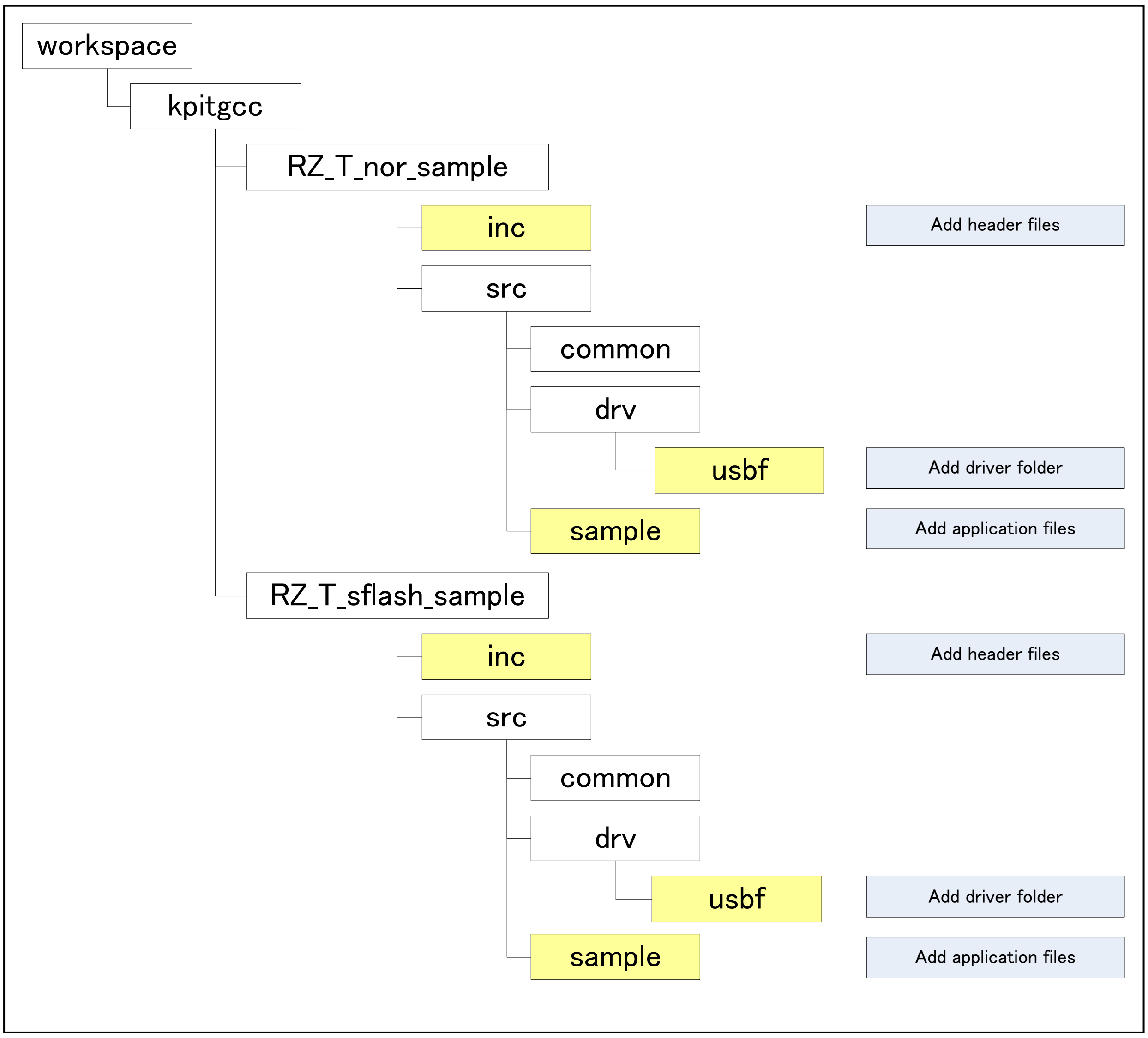
・Add the following files in the “sample” folder.
r_ram_disk_format_data.c
r_usb_pmsc_apl.c
r_usb_pmsc_descriptor.c

・Add the “usbf” folder and the following files “usbf” folder in the “drv” folder.

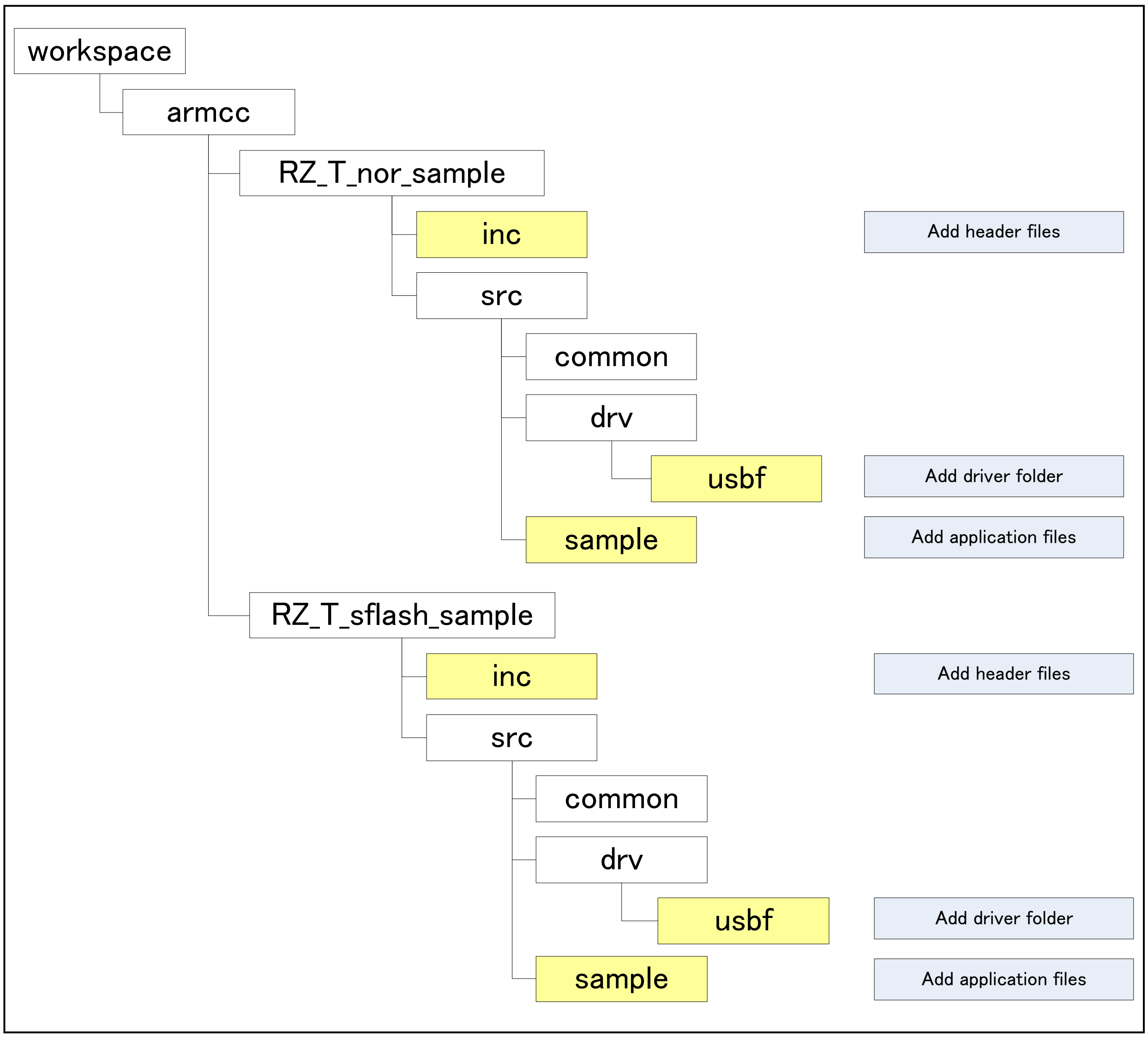
The following is the folder structure of EWARM.



The following is the folder structure of e^2 studio.



The following is the folder structure of DS-5.



Section

Modify the section size of the code area and a data area, and add the following section.

Section name	Address	variable	file
USB_MEDISA	0x48000000	g_ramdisk_mem	r_ram_disk_format_data.c

e^2 studio

e^2 studio sets the section in the configuration screen.
Changes are as follows:
・Fixed address of ".data" section from 0x0007F000 to 0x00040000
・Add section setting of the USB_MEDIA.

Refer to [Project] → [Properties] → [C/C++ Build] → [Settings] → [Sections].

Variable definitions in the code are as follows.

r_ram_disk_format_data.c

```
#ifdef __GNUC__
uint8_t g_ramdisk_mem[RAMDISK_MEDIASIZE] __attribute__ ((section ("USB_MEDIA")));
#endif
```

EWARM

EWARM sets the section in the linker setting file (.icf file).

To the USB_MEDIA to fixed address, adds memory region definition.

```
Place in CS2_region { section USB_MEDIA };
```

Variable definitions in the code are as follows.

r_ram_disk_format_data.c

```
#ifdef __ICCARM__
#pragma location="USB_MEDIA"
uint8_t g_ramdisk_mem[RAMDISK_MEDIASIZE];
#endif
```

DS-5

DS-5 sets the section in the linker setting file (.scatter file).

To the USB_MEDIA to fixed address, adds memory region definition.

```
LOAD_MODULE5 0x48000000    (0x48000000 - 0x48ffffff)
{
        USB_MEDIA 0x48000000  0x8000000
        {
                r_ram_disk_format_data.o(USB_MEDIA)
        }
}
```

Variable definitions in the code are as follows.

r_ram_disk_format_data.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "USB_MEDIA"
uint8_t g_ramdisk_mem[RAMDISK_MEDIASIZE];
#endif
```

Call the USB-BASIC-FW function

Adds the usbf_main() of USB-BASIC-F/W in the main() of "\src\sample\int_main.c".

```
extern void usbf_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBf main */
    usbf_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

Website and Support

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

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

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	Aug 21, 2015	—	First edition issued
1.10	Dec 25, 2015	20	Added Appendix A
1.20	Feb 29, 2016	22,24	Added DS-5 setting
1.30	Dec 07, 2017	—	Corresponds to RZ / T1 initial setting Ver 1.30

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. Handling of Unused Pins

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

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

2. Processing at Power-on

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

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

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

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

RENESAS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Colophon 6.0