RZ/A2M Group

USB Host Mass Storage Class Driver (HMSC)

Introduction

This application note describes USB Host Mass Storage Class Driver (HMSC). This module performs hardware control of USB communication. It is referred to below as the USB-BASIC-F/W. It is referred to below as the HMSC.

Please define “#define USBH0_CFG_HMSC_USE” and “#define USBH1_CFG_HMSC_USE” in r_usbh0_basic_config.h and r_usbh1_basic_config.h.

Target Device

RZ/A2M Group

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 HMSC, when used in combination with the USB-BASIC-F/W, operates as a USB host mass storage class driver (HMSC).

The HMSC comprises a USB mass storage class bulk-only transport (BOT) protocol. When combined with a file system and storage device driver, it enables communication with a BOT-compatible USB storage device.

Note that this software uses the FatFs.

This module supports the following functions.

- Checking of connected USB storage devices (to determine whether or not operation is supported).
- Storage command communication using the BOT protocol.
- Support for SFF-8070i (ATAPI) USB mass storage subclass.

The names of API etc are different between port 0 and port 1.

In this document, API names etc. of port 0 are described as an example.

Table 1.1 Peripheral device used

Peripheral device	Usage
Host PC	output messages from the sample code

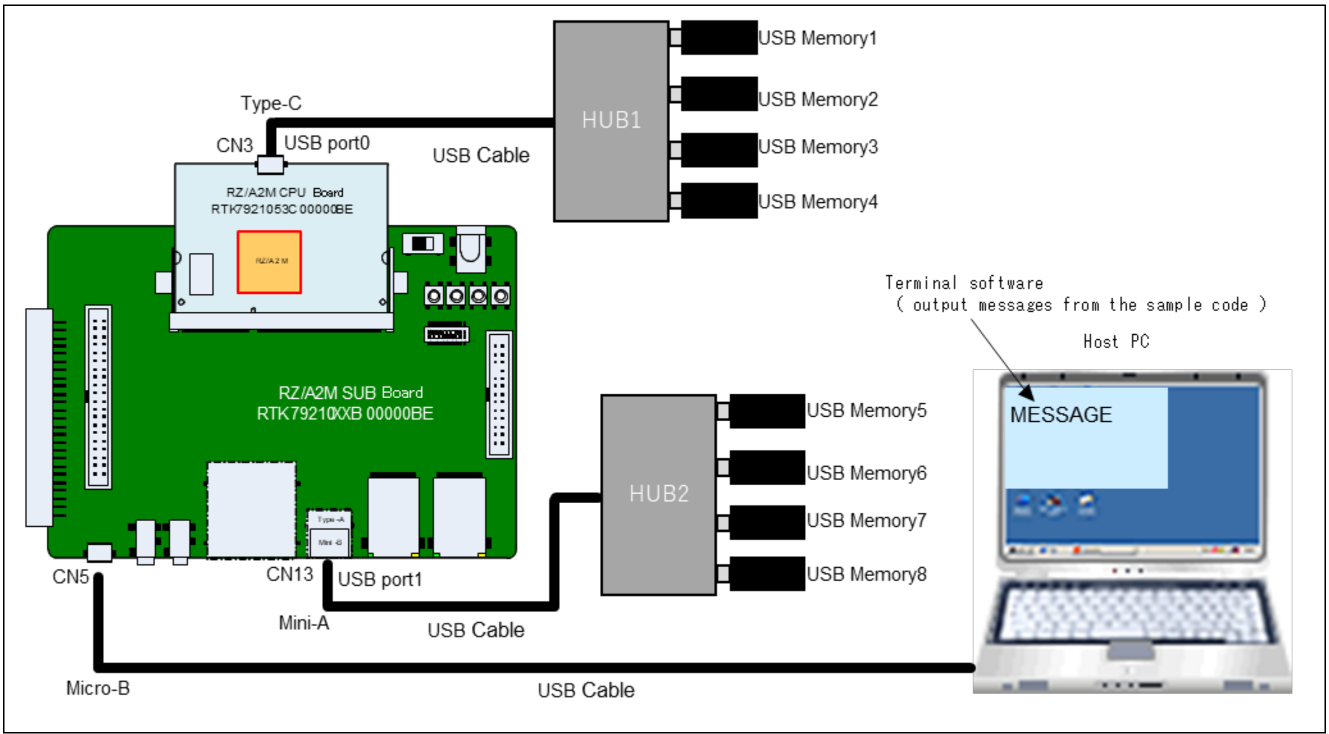

Figure 1-1 Operation check conditions

1.1 Limitations

This module is subject to the following restrictions

1. Structures are composed of members of different types (Depending on the compiler, the address alignment of the structure members may be shifted).
2. Only supported for Logical Unit Number 0 (LUN0).
3. USB storage devices with a sector size of 512 bytes can be connected.
4. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.

1.2 Note

This driver is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.

1.3 Terms and Abbreviations

Abbreviation	Full Form
APL	Application program
ATAPI	AT Attachment Packet Interface
BOT	Mass storage class Bulk Only Transport
CBW	Command Block Wrapper
CSW	Command Status Wrapper
FSI	File System Interface
HCD	Host Control Driver of USB-BASIC-F/W
HMSC	Host Mass Storage Class driver
HMSCD	Host Mass Storage Class Driver unit
HMSDD	Host Mass Storage Device Driver
HUBCD	Hub Class Sample Driver
LUN	Logical Unit Number
LBA	Logical Block Address
MGR	Peripheral device state manager of HCD
SCSI	Small Computer System Interface
Scheduler	Used to schedule functions, like a simplified OS.
Task	Processing unit
USB-BASIC-F/W	USB basic firmware for RZ/A2M
USB	Universal Serial Bus

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 V7.6.0
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. Reference Application Notes

・USB Basic Host Driver Application Note (Document number. R01AN4715EJ0120)

4. Software Configuration

HDCD (Host Device Class Driver) is the all-inclusive term for HMSDD (Host Mass Storage Device Driver) and HMSCD (USB Host Mass Storage Class Driver).

Figure 4-1 shows the HMSC software block diagram, with HDCD as the centerpiece. Table 4-1 describes each module.

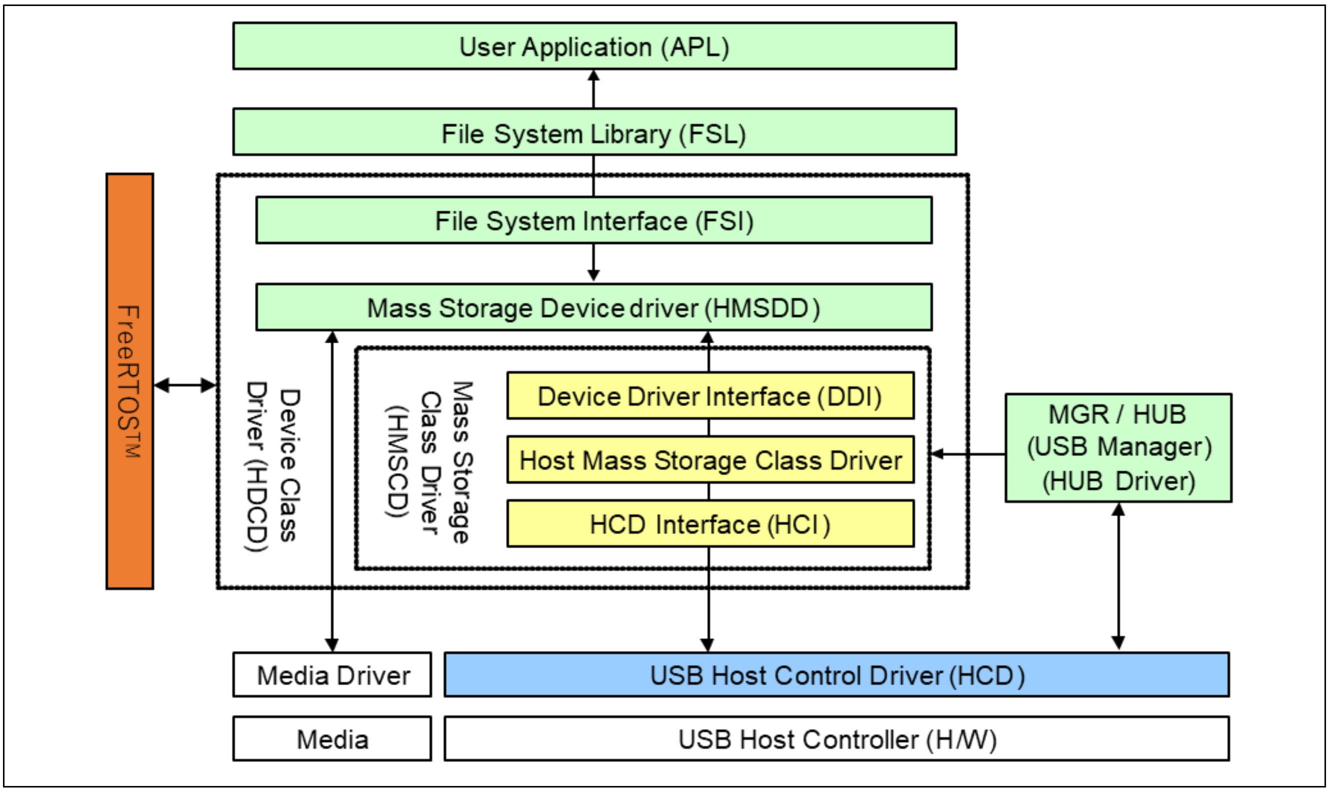

Figure 4-1 Software Block Diagram

Table 4-1 Module

Module	Description
APL	Calls FSL functions to implement storage functionality. Created by the customer to match the system specifications.
FSL	FAT file system with specifications defined by the user.
FSI	FSL-HMSDD interface functions. They should be modified to match FSL.
HMSDD	To be created (modified) by the customer to match the storage media.
DDI	HMSDD-HMSCD interface functions. They should be modified to match the storage media interface of HMSDD.
HMSCD	The USB host mass storage class driver. It appends BOT protocol information to storage commands and sends requests to HCD. It also manages the BOT sequence. The storage commands should be added (modified) by the customer to match the system specifications. SFF-8070i (ATAPI) is supported in the example code.
HCI	HMSCD-HCD interface functions.
MGR/HUB	Enumerates the connected devices and starts HMSCD. Also performs device state management.
HCD	USB host hardware control driver.
Media Driver	Driver for non-USB storage devices.
FreeRTOS™	FreeRTOS™

5. System Resources

The resources used by HMSC are listed below.

Table 3-1 Task Information

Function Name	Task ID	Priority	Description
R_USBH0_HmscTask	USB_HMSC_TSK	USB_PRI_3	Mass storage task
R_USBH0_HmscStrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	HMSDD task

Table 3-2 Mailbox Information

Mailbox Name	Using Task ID	Task Queue	Description
USB_HMSC_MBX	USB_HMSC_TSK	FIFO order	For HMSCD
USB_HSTRG_MBX	USB_HSTRG_TSK	FIFO order	For HMSDD

Table 3-3 Memory Pool Mailbox Information

Memory Pool Name	Task Queue	Memory Block (note)		Description
USB_HMSC_MPL	FIFO order	40byte		For HMSC
USB_HSTRG_MPL	FIFO order	40byte		For HDCD

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

6. Target Peripheral List（TPL）

When using a USB host driver (USB-BASIC-F/W) and device class driver in combination, it is necessary to create a target peripheral list (TPL) for each device driver.

For details on the TPL, refer to “5.4 Target Peripheral List (TPL)” in the Application Note of USB Basic Host Driver (Document No: R01AN4715EJ0120).

7. Accessing USB Storage Devices

After the information acquisition is complete, be able to access the USB storage devices in the API function of FatFs.

When call the API function of FatFs, FSI is called in the file system processing.

HMSDD calls the API function of HMSCD in accordance with the processing.

HMSDC issue a class request and create a USB packet in accordance with BOT protocol.

Under the BOT specification, information is read and written according to logical block addresses. The data size is specified as the number of bytes of information that are read or written.

Figure 7-1 shows the USB storage device access sequence.

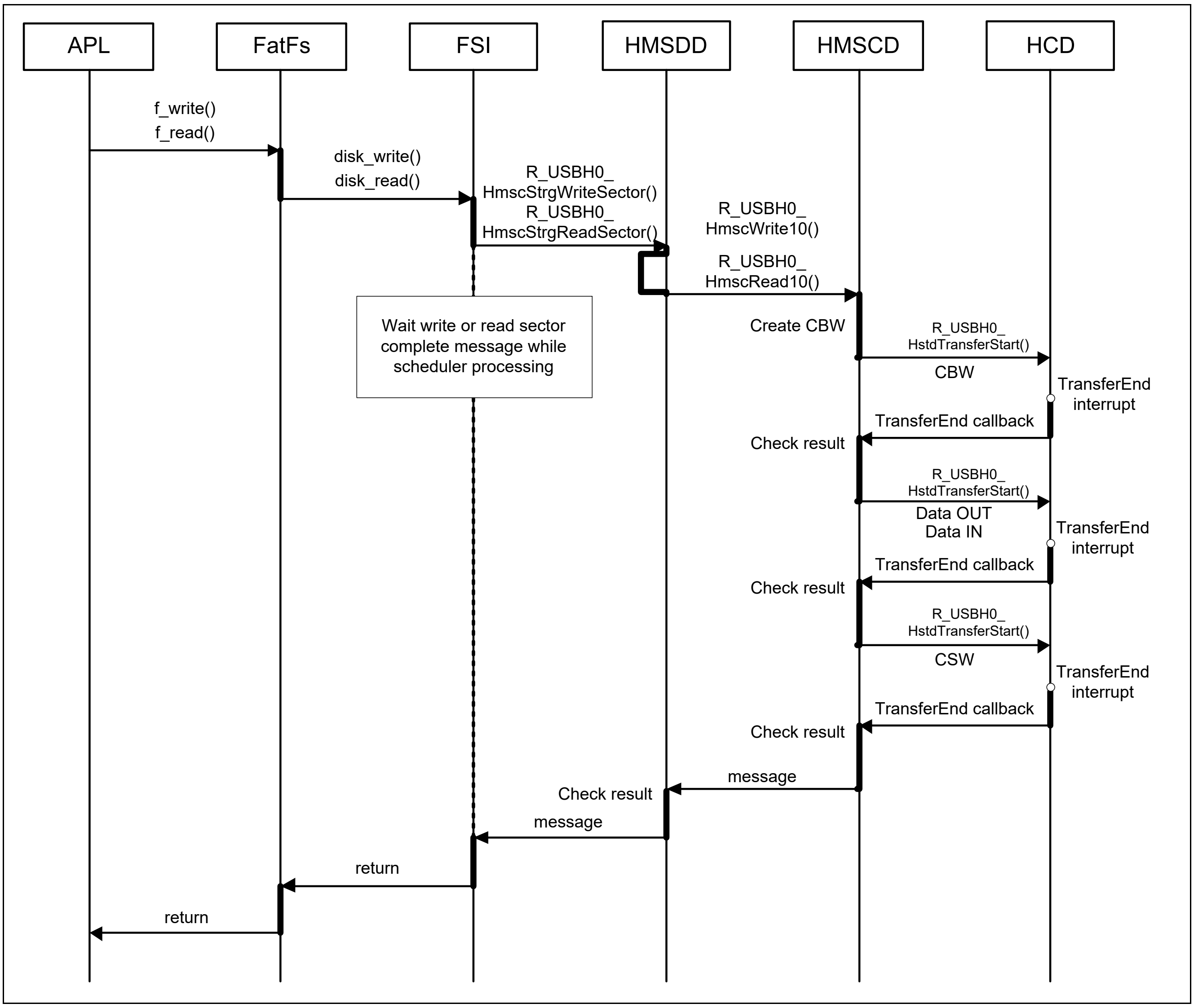

Figure 7-1 USB Storage Device Access Sequence

8. File System Interface (FSI)

8.1 Functions and Features

Create the FSL to match the specifications of the interface.

8.2 FSI API

Table 8-1 lists the functions of the sample FSI.

Table 8-1 FSI Functions

Function Name	Description
disk_read	Read data
disk_write	Write data

9. Host Mass Storage Device Driver (HMSDD)

HMSDD, the "device driver", is called by FAT when using HMSCD. HMSDD selects a storage device depending on a given drive number.

9.1 Limitation

- Maximum 4 USB storage devices can be connected.
- USB storage devices with a sector size of 512 bytes can be connected.
- A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
- Some devices may be unable to be connected (because they are not recognized as storage devices).

9.2 Logical Unit Number (LUN)

If a device does not respond to a GetMaxUnit request and the unit count is undetermined, the device operates with a unit count setting of 0. HMSCD creates USB packets according to the BOT specification. The CBWCB field (storage command) of the data packet is created according to the SCSI specification. The LUN field settings in the bCBWLUN field of the CBW packet and in the storage command are listed in.Table 9-1.

Table 9-1 LUN Field Settings

	bCBWLUN Field	LUN Field in Command
usb_ghmsc_MaxLUN = 0	0	0
usb_ghmsc_MaxLUN = other than 0	Unit number	0

9.3 Changing Logical Block Addresses

Under the BOT specification, information is read and written according to logical block addresses. The data size is specified as the number of bytes of information that are read or written.

The logical block address is calculated from the sector number and offset sector number. The transfer size is calculated from the sector count and sector size.

Logical block address = logical sector number + offset sector number

Transfer size = sector count * sector size

9.4 HMSDD API Function

Table 9-2 lists the function of HMSDD.

Table 9-2 HMSDD functions

Function Name	port	Description
R_USBH0_HmscStrgDriveTask() R_USBH1_HmscStrgDriveTask()	0 1	Gets the storage device information
R_USBH0_HmscStrgDriveSearch() R_USBH1_HmscStrgDriveSearch()	0 1	Searchs the accessible drive
R_USBH0_HmscStrgReadSector() R_USBH1_HmscStrgReadSector()	0 1	Reads data.
R_USBH0_HmscStrgWriteSector() R_USBH1_HmscStrgWriteSector()	0 1	Writes data.
R_USBH0_HmscStrgUserCommand() R_USBH1_HmscStrgUserCommand()	0 1	Issues storage command.

9.4.2 R_USBH0_HmscStrgDriveTask

Storage drive task

Format

void R_USBH0_HmscStrgDriveTask(void)

Argument

— —

Return Value

— —

Description

This API does the processing to get the storage device information by sending SFF-8070i (ATAPI) command to the storage device.

Notes

1. Please call this function from the application program

Example

```
void usb_hapl_mainloop(void)
{
  while(1)
  {
    R_USBH0_CstdScheduler ();

    if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
    {
        R_USBH0_HstdMgrTask();          /* MGR task */
        R_USBH0_HhubTask();             /* HUB task */
        R_USBH0_HmscTask();             /* HMSC Task */
        R_USBH0_HmscStrgDriveTask();    /* HSTRG Task */

    }
                 :
  }
}
```

9.4.4 R_USBH0_HmscStrgDriveSearch

Searchs the accessible drive

Format

uint16_t R_USBH0_HmscStrgDriveSearch(uint16_t addr, usbh0_utr_cb_t complete)

Argument

addr	USB address
complete	Callback function

Return Value

USBH0_OK	Normal end
USBH0_ERROR	Error end

Description

This API checks the follows by sending the class request (GetMaxLun) and SFF-8070i (ATAPI) command to USB device which is specified in the argument (addr).

1. The number of unit of the storage device
2. Accesible the drive

The callback function which is specified in the argument (complete) is called when completing the drive searching.

Notes

1. Please call this function from the class driver or the FAT library I/F function.

Example

```
/* Callback function */
void usb_hmsc_StrgCommandResult( st_usbh0_utr_t *utr )
{
        :
}

void usb_hmsc_SampleAplTask(void)
{
        :
    R_USBH0_HmscStrgDriveSearch(addr, usb_hmsc_StrgCommandResult);
        :
}
```

9.4.6 R_USBH0_HmscStrgReadSector

Read Sector Information

Format

uint16_t	R_USBH0_HmscStrgReadSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

side	Drive number
*buff	Pointer to read data storage area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USBH0_OK	Normal end
USBH0_ERROR	Error end

Description

Reads the sector information of the drive specified by the argument.
An error response occurs in the following cases.

1. When the sector information could not be read successfully from the storage device.

Notes

1. Please call this function from FAT library I/F function.

Example

```
int read_sector(int side, unsigned char *buff, unsigned long secno,
                long seccnt)
{
                :
    error = R_USBH0_HmscStrgReadSector(uint16_t)side, buff, secno, (uint16_t)seccnt,
                trans_byte);

    if( error == USBH0_ERROR )
    {
                :
        return (-1);
    }
    return (0);
}
```

9.4.7 R_USBH0_HmscStrgWriteSector

Write Sector Information

Format

uint16_t	R_USBH0_HmscStrgWriteSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

side	Drive number
*buff	Pointer to write data storage area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USBH0_OK	Normal end
USBH0_ERROR	Error end

Description

Writes the sector information of the drive specified by the argument.
An error response occurs in the following cases.

1. When the sector information could not be read successfully from the storage device.

Notes

1. Please call this function from FAT library I/F function.

Example

```
int write_sector(int side, unsigned char *buff, unsigned long secno,
                long seccnt)
{
               :
    error = R_USBH0_HmscStrgWriteSector((uint16_t)side, buff, secno, (uint16_t)seccnt,
               trans_byte);

    if( error == USBH0_ERROR )
    {
               :
        return (-1);
    }
    return (0);
}
```

9.4.9 R_USBH0_HmscStrgUserCommand

Issue Storage Command

Format

uint16_t R_USBH0_HmscStrgUserCommand(uint16_t side, uint16_t command , uint8_t *buff, usbh0_utr_cb_t complete)

Argument

side	Drive number
command	Command to be issued
*buff	Data pointer
complete	Callback function

Return Value

USBH0_OK	Normal end
USBH0_ERROR	Error end

Description

This function issues the storage command specified by the given argument, to the HMSC driver. The callback function which is specified in the argument (complete) is called when completing the issued storage command. Here are the storage commands supported:

Storage commands	Description
USB_ATAPI_TEST_UNIT_READY	Check status of peripheral device
USB_ATAPI_REQUEST_SENCE	Get status of peripheral device
USB_ATAPI_INQUIRY	Get parameter information of logical unit
USB_ATAPI_MODE_SELECT6	Specify parameters
USB_ATAPI_PREVENT_ALLOW	Enable/disable media removal
USB_ATAPI_READ_FORMAT_CAPACITY	Get formattable capacity
USB_ATAPI_READ_CAPACITY	Get capacity information of logical unit
USB_ATAPI_MODE_SENSE10	Get parameters of logical unit

Notes

Please call this function from the application program and the class driver.

Example

```
/* Callback function */
void  strgcommand_complete(uint16_t data1, uint16_t data2)
{
            :
}

void  usb_smp_task(void)
{
           :
 /* TEST_UNIT_READY 発行 */
     err = R_USBH0_HmscStrgUserCommand(side, USB_ATAPI_TEST_UNIT_READY, buf,
            strgcommand_complete);
           :
}
```

10. USB Mass Storage Class Driver (HMSCD)

10.1 Functions and Features

HMSCD executes storage command communication, if the USB storage devices are ready to operate. The BOT protocol is used, which encapsulates the storage commands as they pass through USB.

MSCD comprises three layers: HMSDD interface (DDI functions), HCD interface (HCI functions), and HMSCD itself.

HMSCD supports storage commands necessary for accessing USB storage devices and sample storage commands.

HMSCD has the following features.

- Support for USB mass storage class BOT.
- Support for SFF-8070i (ATAPI) and SCSI USB mass storage subclasses.
- Sharing of a single pipe for IN/OUT directions or multiple devices.

10.2 Issuing Requests to HMSCD

The interface functions described below (Table 10-4and Table 10-5) are used when accessing USB storage devices.

HMSCD sends notification of results in response to requests from a higher layer in the return value of the registered callback function...

10.3 HMSCD Structures

Table 10-1 and Table 10-2 show the contents of the HMSCD structures.

Table 10-1 st_usbh0_hmsc_cbw_t Structure

	Member Name	Description	Remarks
uint32_t	dcbw_signature	CBW Signature	0x55534243: USBC
uint32_t	dcbw_tag	CBW Tag	Tag corresponding to CSW
uint8_t	dcbw_dtl_lo	CBW Data Transfer Length	Data length of transmit/receive data
uint8_t	dcbw_dtl_ml		
uint8_t	dcbw_dtl_mh		
uint8_t	dcbw_dtl_hi		
uint8_t	bmcbw_flags	CBW Direction	Data transmit/receive direction
uint8_t	bcbw_lun	Logical Unit Number	Unit number
uint8_t	bcbw_cb_length	CBWCB Length	Command length
uint8_t	cbw_cb[16]	CBWCB	Command block

Table 10-2 st_usbh0_hmsc_csw_t Structure

	Member Name	Description	Remarks
uint32_t	dcsw_signature	CSW Signature	0x55534253: USBS
uint32_t	dcsw_tag	CSW Tag	Tag corresponding to CBW
uint8_t	dcsw_data_residue_lo	CSW DataResidue	Data length used
uint8_t	dcsw_data_residue_ml		
uint8_t	dcsw_data_residue_mh		
uint8_t	dcsw_data_residue_hi		
uint8_t	bcsw_status	CSW Status	Command status

10.4 USB Host Control Driver Interface (HCI) Functions

HCI is the interface function between HMSCD and HCD.

HCI uses the HMSCD area to send and receive messages.

Note that two devices cannot be enumerated (or accessed) simultaneously.

10.5 Device Driver Interface (DDI) Functions

DDI is the interface function between HMSDD and HMSCD.

DDI functions comprise the HMSCD start function, end function, check connected device function, and sample storage command functions.

10.6 HMSC API

Application programming interface description for HMSCD

Table 10-3 HMSCD Functions

	Function Name	port	Description
1	R_USBH0_HmscGetDevSts() R_USBH1_HmscGetDevSts()	0 1	Returns HMSCD operation state.
2	R_USBH0_HmscTask() R_USBH1_HmscTask()	0 1	A task for HMSCD
3	R_USBH0_HmscDriverStart() R_USBH1_HmscDriverStart()	0 1	HMSC driver start processing.
4	R_USBH0_HmscAllocDrvno() R_USBH1_HmscAllocDrvno()	0 1	Allocates the drive number.
5	R_USBH0_HmscFreeDrvno() R_USBH1_HmscFreeDrvno()	0 1	Frees the drive number
6	R_USBH0_HmscRefDrvno() R_USBH1_HmscRefDrvno()	0 1	Refers the drive number

Table 10-4 HCI Functions

	Function Name	port	Description
1	R_USBH0_HmscGetMaxUnit() R_USBH1_HmscGetMaxUnit()	0 1	Get_MaxUnit request execution.
2	R_USBH0_HmscMassStorageReset() R_USBH1_HmscMassStorageReset()	0 1	MassStorageReset request execution.

Table 10-5 DDI Functions

	Function Name	port	Description
1	R_USBH0_HmscClassCheck() R_USBH1_HmscClassCheck()	0 1	Checks the descriptor table of the connected device to determine whether or not HMSCD can operate.
2	R_USBH0_HmscRead10() R_USBH1_HmscRead10()	0 1	Issues the READ10 command.
3	R_USBH0_HmscWrite10() R_USBH1_HmscWrite10()	0 1	Issues the WRITE10 command.

10.6.2 R_USBH0_HmscGetDevSts

Returns HMSCD operation state

Format

uint16_t R_USBH0_HmscGetDevSts(uint16_t drvno)

Argument

drvno Drive number

Return Value

USBH0_TRUE (Attach)
USBH0_FALSE (Detach)

Description

Returns the HMSCD operation state.

Note

1. Please call this function from the user application program or the class driver.

Example

```
void  usb_smp_task()
{
    :
  /* Checking the device state */
  if(R_USBH0_HmscGetDevSts(drvno) == USBH0_FALSE)
  {
    /* Detach processing */

  }
    :
}
```

10.6.4 R_USBH0_HmscTask

Host Mass Storage Class task

Format

void R_USBH0_HmscTask(void)

Argument

— —

Return Value

— —

Description

This function is a task for HMSCD.
This function controls BOT.

Note

1. Please call this function from a loop that executes the scheduler processing for non-OS operations.
2. Please refer to the chapter "Operation Flow in Static State" in the Basic F/W application note for more information about this loop.

Example

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Scheduler processing */
  R_USBH0_CstdScheduler();
  /* Checking flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
        :
    R_USBH0_HmscTask();
        :
  }
    :
}
```

10.6.6 R_USBH0_HmscDriveStart

HMSC driver start

Format

void R_USBH0_HmscDriverStart(void)

Argument

— —

Return Value

— —

Description

This function sets the priority of HMSC driver task.
The sent and received of message are enable by the priority is set.

Note

1. Call this function from the user application program during initialization.

Example

```
void usb_hstd_task_start( void )
{
     :
  R_USBH0_HmscDriverStart();     /* Host Class Driver Task Start Setting */
     :
}
```

10.6.7 R_USBH0_HmscAllocDrvno

Allocates the driver number

Format

uint16_t R_USBH0_HmscAllocDrvno(uin16_t *p_side, uint16_t devadr)

Argument

*p_side	drive number (output)
devadr	Device address of MSC device

Return Value

USBH0_OK	Success
USBH0_ERROR	Failure

Description

This function allocates the drive number to the connected MSC device.

Notes

1. The user can specifies the drive number which is allocated by this API in the argument of FAT API.

Example

```
void  usb_smp_task(void)
{
    uint16_t    *side;
    uint16_t    devadr;
    uint16_t    err;
            :
    /* Allocates the drive number */
    drvno = R_USBH0_HmscAllocDrvno(side, devadr);
            :
}
```

10.6.9 R_USBH0_HmscFreeDrvno

Frees the driver number

Format

uint16_t R_USBH0_HmscFreeDrvno(uint16_t side)

Argument

side Drive number

Return Value

USBH0_OK	Success
USBH0_ERROR	Failure

Description

This function frees the drive number which is specified by the argument.

Notes

—

Example

```
void  usb_smp_task(void)
{
    uint16_t    devno;
    uint16_t    err;
            :
    /* Frees the drive number */
    err = R_USBH0_HmscFreeDrvno(drvno);
    if(err == USBH0_ERROR)
    {
       /* Error processing */
    }
            :
}
```

10.6.11 R_USBH0_HmscRefDrvno

Refers the driver number

Format

uint16_t	R_USBH0_HmscRefDrvno(uint16_t *p_side, uint16_t devadr)

Argument

*p_side	drive number (output)
devadr	Device address

Return Value

USBH0_OK	Success
USBH0_ERROR	Failure

Description

This function refers the drive number based on USB module number and the device address which are specified by the argument.

Notes

—

Example

```
void  usb_smp_task(void)
{
    uint16_t    *side;
    uint16_t    devadr;
    uint16_t    err;
            :
    /* Frees the drive number */
    err = R_USBH0_HmscRefDrvno(side, devadr);
    if(err == USBH0_ERROR)
    {
       /* Error processing */
    }
            :
  }
```

10.6.13 R_USBH0_HmscGetMaxUnit

Issue GetMaxLUN request.

Format

usbh0_er_t R_USBH0_HmscGetMaxUnit(uint16_t addr, usbh0_utr_cb_t complete)

Argument

addr	Device address
complete	Callback function

Return Value

USBH0_OK	GET_MAX_LUN issued
USBH0_ERROR	GET_MAX_LUN not issued

Description

This function issues the GET_MAX_LUN request and gets the maximum storage unit count. The callback function which is specified in the argument (complete) is called when completing this request.

Note

1. Please call this function from the user application program or the class driver.

Example

```
void  usb_smp_task()
{
  usbh0_er_t  err;
    :
  /* Getting Max unit number */
  err = R_USBH0_HmscGetMaxUnit(addr, usb_hmsc_StrgCheckResult);
  if(err == USBH0_ERROR)
  {
    /* Error processing */
  }
    :
}
```

10.6.15 R_USBH0_HmscMassStorageReset

Issue Mass Storage Reset request.

Format

usbh0_er_t R_USBH0_HmscMassStorageReset(uint16_t drvnum, usbh0_utr_cb_t complete)

Argument

drvnum	Drive number
complete	Callback function

Return Value

USBH0_OK	MASS_STORAGE_RESET issued
USBH0_ERROR	MASS_STORAGE_RESET not issued

Description

This function issues the MASS_STORAGE_RESET request and cancels the protocol error.
The callback function which is specified in the argument (complete) is called when completing this request.

Note

1. Please call this function from the user application program or the class driver.

Example

```
void   usb_smp_task()
{
  usbh0_er_t     err;
    :
  /* Cancel the protocol error */
  err = R_USBH0_HmscMassStorageReset(drvnum, usb_hmsc_CheckResult);
  if(err == USBH0_ERROR)
  {
    /* Error processing */
  }
    :
}
```

10.6.17 R_USBH0_HmscClassCheck

Compare connected device with interface descriptor

Format

void	R_USBH0_HmscClassCheck(uint16_t **table)

Argument

**table	Not used

Return Value

—	—

Description

Checks the device count and drive count, and analyzes the interface descriptor table. Confirms that the items listed below match HMSCD, and reads the serial number if the operation is possible. Updates the pipe information table with information from the bulk endpoint descriptor table (endpoint address, max. packet size, etc.).

Interface descriptor information check
- bInterfaceSubClass = USB_ATAPI / USB_SCSI
- bInterfaceProtocol = USB_BOTP
- bNumEndpoint > USB_TOTALEP

String descriptor information check

Serial number of 12 or more characters (warning indication in case of error)

Endpoint Descriptor information check
- bmAtributes = 0x02 (bulk endpoint required)
- bEndpointAdress (endpoints required for both IN and OUT)

One of the following check results is returned as Table [3].
- USB_DONE: HMSCD operation possible
- USB_ERROR: HMSCD operation not possible

Note

1. In application program, register this API as the callback function by setting this API in the member *classcheck*.
2. The maximum connectable storage device count is defined by USB_MAXSTRAGE. (Refer to r_usbh0_hmsc_define.h)

Example

```
void   usb_hapl_registration()
{
    :
  /* Driver check */
  driver.classcheck = &R_USBH0_HmscClassCheck;
    :
}
```

10.6.19 R_USBH0_HmscRead10

Issue a READ10 command

Format

uint16_t	R_USBH0_HmscRead10(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

side	Drive number
*buff	Read data area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USBH0_OK

Description

Creates and executes the READ10 command.
When a command error occurs, the REQUEST_SENSE command is executed to get error information.

Note

1. Please call this function from the user application program or the class.driver.

Example

```
void   usb_smp_task()
{
  uint32_tresult;
    :
  /* Issuing READ10 */
  R_USBH0_HmscRead10(side, buff, secno, seccnt, trans_byte);
    :
}
```

10.6.21 R_USBH0_HmscWrite10

Issue a WRITE10 command

Format

uint16_t R_USBH0_HmscWead10(uint16_t side, uint8_t *buff, uint32_t secno,
uint16_t seccnt, uint32_t trans_byte)

Argument

side	Drive number
*buff	Write data area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USBH0_OK

Description

Creates and executes the WRITE10 command.
When a command error occurs, the REQUEST_SENSE command is executed to get error information.

Note

1. Please call this function from the user program or the class driver.

Example

```
void  usb_smp_task()
{
    :
  /* Issuing WRITE10 */
  R_USBH0_HmscWrite10(side, buff, secno, seccnt, trans_byte);
    :
}
```

11. Sample Application

11.1 Application Specifications

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

1. When the application is started, the following message is output on the terminal software.
   SAMPLE> USB HMSC Application
   0 : READ loop mode
   1 : READ stop mode
2. The APL performs the enumeration and drive recognition processing on an MSC device.
3. Get the drive information of the MSC device.
4. Generate a text file 'SAMPLE? .txt' with 512 'a' written on the MSC device and read the generated file once. The? In the file name is one of 0-7.
5. Enter "0" from the terminal software here to continue reading the generated file.
   When "1" is entered, file reading ends.
6. By using a USB Hub, the APL can perform the above processing 1 through 4 on up to four MSC devices.

[Note]

1. If the MSC device where the file ’ SAMPLE? .txt’ is stored is connected, that file is overwritten.
2. For setting the terminal software, see “Table 1.1 Peripheral device used”.

11.2 Application Processing

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

11.2.1 Initial Setting

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

11.2.2 Main Loop

After the USB driver initial settings, call the scheduler (R_USBH0_CstdScheduler()) from the main routine of the application. Calling R_USBH0_CstdScheduler() from the main routine causes a check for events. If there is an event, a flag is set to inform the scheduler that an event has occurred. After calling R_USBH0_CstdScheduler(), call R_USBH0_CstdCheckSchedule() to check for events. Also, it is necessary to run processing at regular intervals to get events and perform the appropriate processing.

In FreeRTOS, use four Non OS functions called in the Non OS main loop as OS tasks, so call R_USBH0_Init () with BSP_CFG_RTOS_USED = 1.
Therefore, when BSP_CFG_RTOS_USED = 1, it is not necessary to call the four Non OS functions in the main loop.
The scheduler R_USBH0_CstdScheduler () sends a message to each task function regardless of Non OS or OS, and controls task processing.

```
  void usb_main(void)
  {
    while(1)  // Main routine
    {
      // Confirming the event and getting (Note 1)
      R_USBH0_CstdScheduler();
#if(BSP_CFG_RTOS_USED == 0)
      // Judgment whether the event is or not
      if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
      {
          R_USBH0_HstdMgrTask();     // MGR task
          R_USBH0_HhubTask();// HUB task (Note 3)                    (Note 2)
          R_USBH0_HmscTask();// MSC task
          R_USBH0_HmscStrgDriveTask();      // STRG driver task
      }
#endif /* (BSP_CFG_RTOS_USED == 0) */
      usbh0_msc_main();      // User application program
    }
  }
```

[Note]

1. If, after getting an event with R_USBH0_CstdScheduler () and before running the corresponding processing, R_USBH0_CstdScheduler () is called again and gets another event, the first event is discarded. After getting an event, always call the corresponding task to perform the appropriate processing.
2. Be sure to describe these processes in the OSLess main loop for the application program.
3. It is only necessary to call this function when the HUB will be used.

11.2.3 APL

APL is managed by the state transition.

Table 11-1 shows list of states.

Table 11-1 List of States

State	Description
STATE_ATTACH	Wait attach
STATE_DRIVE	Search drive
STATE_READY	Mount drive
STATE_WRITE	File Write
STATE_READ	File Read
STATE_COMPLETE	Processing completion
STATE_ERROR	Error occurred

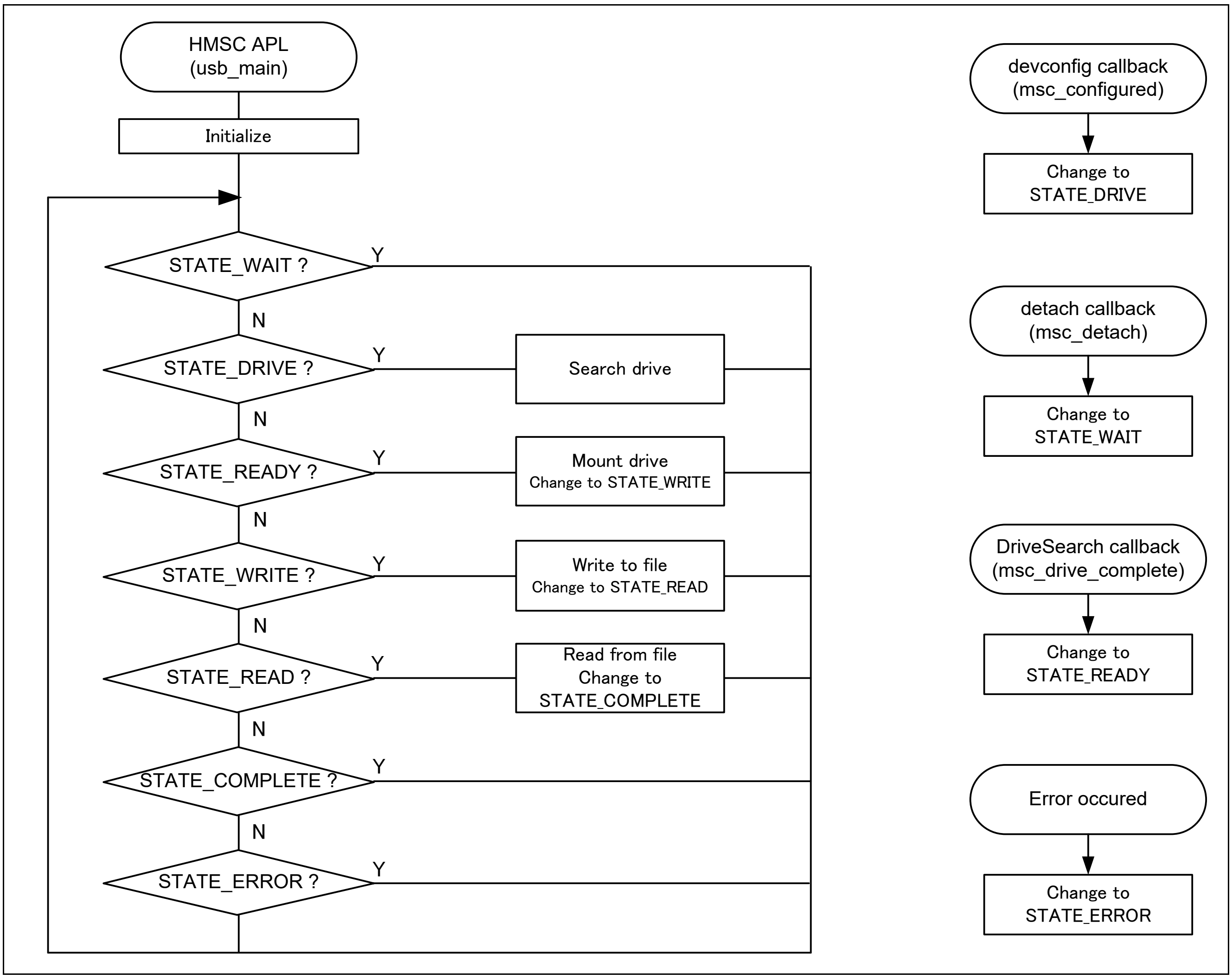

Figure 11-1 Main Loop flowchart

11.2.4 State Management

An overview of the processing associated with each state is provided below.

1) Wait Attach (STATE_WAIT)

== Outline ==

In this state, wait for the MSC device attach. When the enumeration is complete, then changes the state to STATE_DRIVE.

== Description ==

1. Initialization function sets the state to STATE_WAIT.
2. Continue to STATE_WAIT until the MSC device is connected.
3. When the MSC device is connected to enumeration is complete, The callback function usbh0_msc_configured() is specified in the member devconfig of structure usbh0_cb_t is called from the USB driver.
4. Changes the state to STATE_DRIVE.

2) Search drive (STATE_DRIVE)

== Outline ==

In this state, make the drive search process of the connected MSC device and change the state to STATE_READY.

== Description ==

1. Check the drive recognition flag variable usbh0_drive_search_lock. Start the process if it's off.
2. Turn on the usbh0_drive_search_lock.
3. Call R_USBH0_HmscStrgDriveSearch (), transmit class request GetMaxLUN and storage command to MSC device, and make the drive search process.
4. When completion of drive search process, R_USBH0_HmscStrgDriveSearch () callback function was registered in usbh0_msc_drive_complete() is called.
5. Change the state to STATE_READY.

3) Mount drive (STATE_READY)

== Outline ==

In this state, mount the recognized drive and change the state to STATE_WRITE.

== Description ==

1. Call f_mount(), mount in the recognized drive number.
2. Changes the state to STATE_WRITE.

4) File Write (STATE_WRITE)

== Outline ==

In this state, write the file to the connected MSC device and change the state to STATE_READ.

== Description ==

1. Call f_open(), Open the file in create and write mode.
2. Call f_write(), create the file of 512bytes of all 'a' (SAMPLE?.txt). ? in the file name corresponds to the drive number. For example, in the case of drive 1, file name is SAMPLE1.txt.
3. Call f_close(), close the file.

4. Changes the state to STATE_READ.

5) File Read (STATE_READ)

== **Outline** ==

In this state, read the file from the connected MSC device and change the state to STATE_COMPLETE.

== **Description** ==

1. Call f_open(), open the file in read mode.
2. Call f_read(), read the file (SAMPLE?.txt).
3. Check whether 512 bytes of data of all ‘a’
4. Call f_close(), close the file.
5. Changes the state to STATE_COMPLETE.

6) Processing completion (STATE_COMPLETE)

== **Outline** ==

When the processing of the sample application is normally finished, will be in this state.

== **Description** ==

None processing.

7) Error occurred (STATE_ERROR)

== **Outline** ==

When the processing of the sample application is abnormally terminated, will be in this state.

== **Description** ==

None processing.

8) Detach processing (STATE_DETACH)

When the connected MSC device is disconnected, the USB driver calls the callback function usbh0_msc_detach(). This callback function performs to initialize variables and unmount drive and change the state to STATE_WAIT. The callback function usbh0_msc_detach() is the function set in the member devdetach of the structure usbh0_cb_t.

12. 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.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.15.19	-	First edition issued
1.10	May.17.19	5	Table 2.1 Operation Confirmation Conditions(1/2) Remove compiler option "-mthumb-interwork"
1.20	Dec.17.19	5	Table 2.1 Operation Confirmation Conditions(1/2) Change compiler option "-g3" to "-None" Support both FreeRTOS / OSLess

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

Trademarks

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/.