RENESAS

APPLICATION NOTE

SH7216 Group

USB Function Module: USB Mass Storage Class

Introduction

This application note describes how to use the USB function module of the SH7216 and examples of creation of firmware conforming to USB Mass Storage Class.

The contents of this application note and the software are provided for describing application examples of the USB function module, but not for ensuring the contents.

Target Device

SH7216

Contents

1. Preface

1.1 Specifications

This application note describes how to use the USB function module of the SH7216 and examples of creation of firmware conforming to USB Mass Storage Class.

1.2 Modules Used

- Interrupt controller (INTC)
- Pin function controller (PFC)
- USB function module (USB)

1.3 Applicable Conditions

• MCU	SH7216
• Operating frequency	Internal clock: 200 MHz Bus clock: 100 MHz Peripheral clock: 50 MHz
• Integrated development environment	High-performance Embedded Workshop Ver.4.05.00 (from Renesas Technology Corp.)
• C compiler	SuperH RISC engine Family C/C++ Compiler Package Ver.9.02 Release00, available from Renesas Technology
• Compiler options	Default settings of the High-performance Embedded Workshop (-cpu=sh2afpu -pic=1 -object="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo)

1.4 Related Application Note

- SH7216 Group Application Note: USB Function Module: USB HID Class (REJ06B0898)

2. Overview

This program performs control transfer, bulk transfer, and processing for Mass Storage Class commands that use the USB function module (USB).

The features of the USB Function Module contained in the SH7216 are listed below.

- Automatic processing of USB protocol
- Automatic processing of USB standard commands for endpoint 0 (some commands need to be processed through the firmware)
- Full-speed transfer supported
- Interrupt request: Various interrupt signals needed for USB transmission and reception are generated.
- Clock: External input clock generated by the USB oscillator (48 MHz)
- Low power consumption mode provided.
- Internal USB transceiver
- Endpoint configurations: The configurations are indicated in table 1.

Table 1 Endpoint Configurations

Endpoint Name	Name	Transfer Type	Max. Packet Size	FIFO Buffer Capacity	DMA Transfer
Endpoint 0	EP0s	Setup	8 bytes	8 bytes	—
	EP0i	Control In	16 bytes	16 bytes	—
	EP0o	Control Out	16 bytes	16 bytes	—
Endpoint 1	EP1	Bulk-in	64 bytes	64 × 2 (128) bytes	Possible
Endpoint 2	EP2	Bulk-out	64 bytes	64 × 2 (128) bytes	Possible
Endpoint 3	EP3	Interrupt In	16 bytes	16 bytes	—
Endpoint 4	EP4	Bulk-in	64 bytes	64 × 2 (128) bytes	Possible
Endpoint 5	EP5	Bulk-out	64 bytes	64 × 2 (128) bytes	Possible
Endpoint 6	EP6	Interrupt In	16 bytes	16 bytes	—
Endpoint 7	EP7	Bulk-in	64 bytes	64 bytes	—
Endpoint 8	EP8	Bulk-out	64 bytes	64 bytes	—
Endpoint 9	EP9	Interrupt In	16 bytes	16 bytes	—

Figure 1 shows an example of a system configuration.

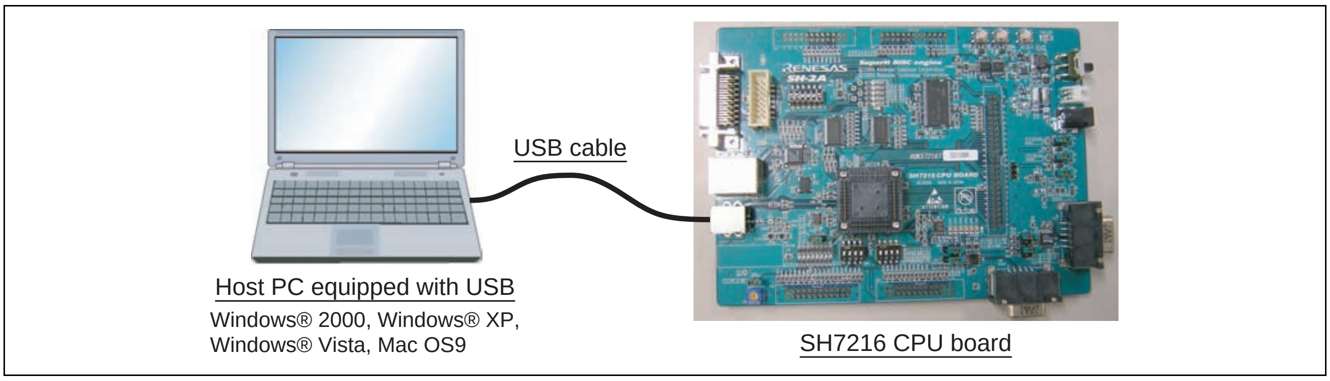

Figure 1 System Configuration Example

This system is configured of the SH7216 CPU Board made by Renesas Technology Corp. and a PC containing Windows® 2000/Windows® XP/Windows® Vista or Mac OS9 operating system.

By connecting the host PC and the SH7216 CPU Board through USB, the SDRAM in the SH7216 CPU Board can be accessed as a RAM disk, enabling data in the SDRAM of the SH7216 CPU Board to be stored in and loaded from the host PC.

It is also possible to use the USB Mass Storage Class (Bulk-Only Transport) device driver that comes as an accessory with the operating systems listed above.

This system offers the following features.

1. The sample program can be used to evaluate the USB module of the SH7216.
2. The sample program supports USB control transfer and bulk transport.
3. This system can be debugged with the E10A (USB Emulator).
4. Additional programs can be created to support interrupt transfer.*

Note: * Interrupt transfer programs are not provided, and will need to be created by the user.
The SH7216 does not support isochronous transfer.

3. Overview of the USB Mass Storage Class (Bulk-Only Transport)

This section describes the USB Mass Storage Class (Bulk-Only Transport).

We hope that it will provide a convenient reference for use when developing USB storage-related systems. For more detailed information on standards, please see (3) and (4) of the section 6 "Documents for Reference".

3.1 USB Mass Storage Class

USB Mass Storage Class is a class of standards that apply to large-scale memory (storage) devices that are connected to a host PC and handle reading and writing of data.

In order to let the PC know that a function is in this class, a value of H'08 must be entered in the bInterface Class field of the Interface Descriptor. Furthermore, USB Mass Storage Class must tell the host the serial number using the String Descriptor. Unicode 000000000001 is returned in this sample program.

When transferring data between the host PC and the function, four transport methods defined by the USB are used (control transfer, bulk transport, interrupt transfer, and isochronous transfer).

Protocol codes determine the transport method and how it is used.

In USB Mass Storage Class, there are two types of data transport protocols:

- USB Mass Storage Class Bulk-Only Transport
- USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport

As its name indicates, USB Mass Storage Class Bulk-Only Transport is a data transport protocol that only uses bulk transport.

USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport is a data transport protocol that uses control transfer, bulk transport, and interrupt transfer. CBI Transport is further subdivided into a data transport protocol that uses interrupt transfer, and one that does not use interrupt transfer.

The sample programs provided here use USB Mass Storage Class Bulk-Only Transport as the data transport protocol.

When the host PC uses a device in order to load and save data, instructions (commands) are provided by the host PC to the function. The function then executes those commands to load and save data. The commands sent by the host PC to the function are defined in the form of sub-class code.

3.2 Sub-Class Code

Sub-class codes are values that indicate the command format sent from the host PC to a function by means of command transport. There are seven types of command formats, described in table 2.

Table 2 Sub-Class Code

Sub-Class Code	Command Standards
H'01	Reduced Block Commands (RBC), T10/1240-D
H'02	Attachment Packet Interface (ATAPI) for CD-ROMs. SFF-8020i, Multi-Media Command Set 2 (MMC-2)
H'03	Attachment Packet Interface (ATAPI) for Tape. QIC-157
H'04	USB Mass Storage Class UFI Command Specification
H'05	Attachment Packet Interface (ATAPI) for Floppies. SFF-8070i
H'06	SCSI Primary Commands –2 (SPC-2), Revision 3 or later

In order to tell the host PC the command format supported by the device, a sub-class code value must be entered in the bINterface SubClass field of the Interface Descriptor.

The sample programs used here use a sub-class code value of H'06, which indicates the SCSI Primary Commands.

3.3 Bulk-Only Transport

With Bulk-Only Transport, data is transferred between the host PC and a function using bulk data transport only.

Bulk transport can be divided into two types, depending on the direction in which the data is sent. If data is sent from the host controller to the function, bulk-out transport is used. If data is being sent to the host controller from the function, bulk-in transport is used.

With Bulk-Only Transport, a combination of bulk-out transport and bulk-in transport determined in advance is used to transfer data between the host and the function. Bulk-Only Transport always uses the combination of bulk transports shown in figure 2. These bulk transports have different meanings, but they are handled as stages (transports).

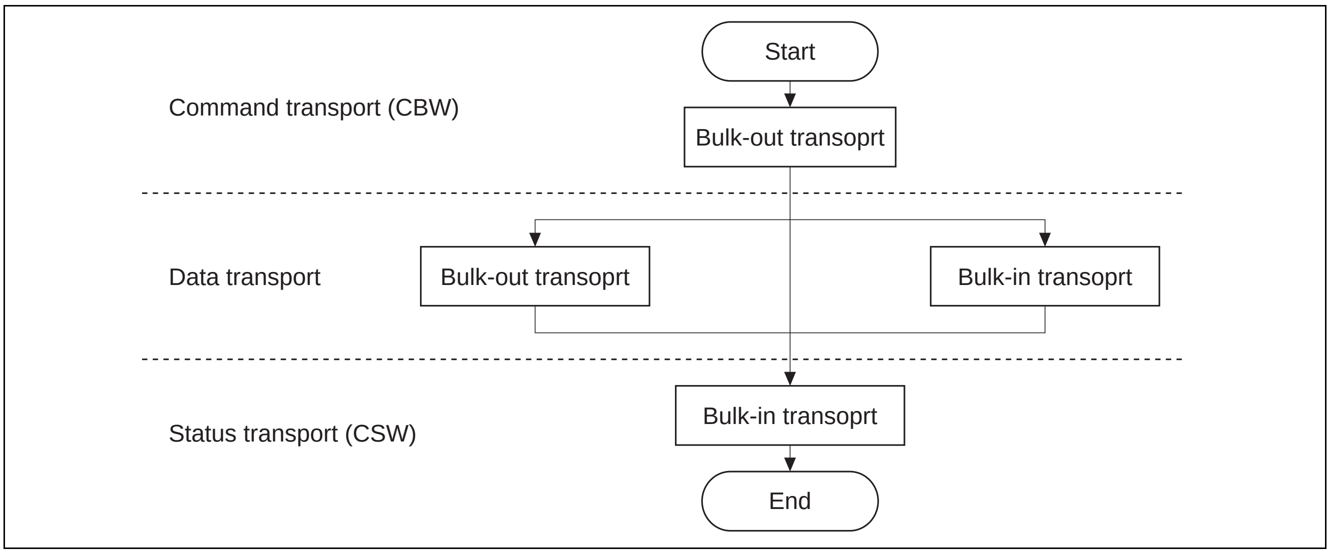

Figure 2 Relationship between Transfer Methods and Transports

In order to tell the host PC that the Bulk-Only Transport protocol is being used, a value of H'50 must be entered in the bInterface Protocol field of the Interface Descriptor.

3.3.1 Command Transport

In command transport, commands are sent from the host PC to the function using bulk-out transport. This command packet is defined as the Command Block Wrapper (CBW), and Bulk-Only Transport must always begin with the CBW.

The CBW is sent from the host PC as a 31-byte packet, using bulk-out transport.

It is sent using the format shown in table 3.

Table 3 Command Transport Formats

	7	6	5	4	3	2	1	0
H'00-H'03	dCBWSignature							
H'04-H'07	dCBWTag							
H'08-H'0B	dCBWDataTransferLength							
H'0C	bmCBWFlags							
H'0D	Reserved (0)				bCBWLUN			
H'0E	Reserved (0)			bCBWCBLength				
H'0F-H'1E	CBWCB							

The fields are explained below.

- dCBWSignature:
  This field identifies the data packet as a CBW. The value is 43425355h (Little Endian).
- dCBWTag:
  This is the command block tag. It is used to connect the CSW with its corresponding CBW, and is specified by the host PC.
- dCBWDataTransferLength:
  This is the length of the data planned for transport. If this is 0, no data transport exists.
- bmCBWFlags:
  If bit 7 of this field is 0, data is transported using bulk-out transport, and if it is 1, bulk-in transport is used. Bits 0 to 6 are fixed at 0.
- bCBWLUN:
  This is the Logical Unit Number of the device sending the command block.
- bCBWCBLength:
  This indicates the number of valid bytes for the next CBWCB field.
- CBWCB:
  This field stores the command block to be executed by the function. The command that the host PC wants to execute (the SCSI command in this sample program) is entered in this field.

3.3.2 Status Transport

Status transport is used to send the results of command execution from the function to the host PC, using bulk-in transport.

This status packet is defined by the Command Status Wrapper (CSW). Bulk-Only Transport must always end with the CSW.

The CSW is sent to the host as a 13-byte packet, using bulk-in transport.

It is sent in the format shown in table 4.

Table 4 Status Transport Formats

	7	6	5	4	3	2	1	0
H'00-H'03	dCSWSignature							
H'04-H'07	dCSWTag							
H'08-H'0B	dCSWDataResidue							
H'0F-H'1E	bCSWStatus							

The fields are explained below.

- dCSWSignature:
  This field identifies the data packet as the CSW. The value is H'53425355 (Little Endian).
- dCSWTag:
  This is the command block tag. It ties the CBW to the CSW, and the same value is entered here as that of the dCBWTag field of the CBW.
- dCSWDataResidue:
  This reports any differences in the value of the CBW dCBWDataTransferLength and the actual amount of data processed by the function.
- bCSWStatus:
  This indicates whether or not a command has been successfully executed. If the command was executed successfully, the function sets H'00 in this field.
  Any value other than 0 indicates that the command was not executed successfully. Error values are as follows: H'01 indicates a failed command, and H'02 indicates a phase error.

3.3.3 Data Transport

Data transport is used to transfer data between the host PC and the function. For example, with the Read/Write command, the actual data of the various storage sectors is sent using data transport.

Data transport is configured of multiple bus transactions.

Data transfers carried out using data transport use either bulk-out or bulk-in transport. The bmCBWFlags field of the CBW data determines which type of transport is used.

(1) Data transport (bulk-out transport)
This section explains how data is transferred when bulk-out data transport is used.
This status is set if bit 7 of the bmCBWFlags field of the CBW data is 0, and the dCBWDataTransferLength field of the CBW data is not 0.
Here, the function receives the anticipated length of the data indicated by the dCBWDataTransferLength field of the CBW data. The data transferred at this point is needed when the SCSI command specified by the CBWCB field of the CBW data is executed.

(2) Data transport (bulk-in transport)
This section explains how data is transferred when bulk-in data transport is used.
This status is set if bit 7 of the bmCBWFlags field of the CBW data is 1, and the dCBWDataTransferLength field of the CBW data is not 0.
Here, the anticipated length of the data indicated by the dCBWDataTransferLength field of the CBW data is sent to the host PC. The data transferred at this point is the result produced when the SCSI command specified by the CBWCB field of the CBW data was executed.

3.4 Class Commands

Class commands are commands that are defined by the various USB classes. They use control transfer.

When USB Mass Storage Class Bulk-Only Transport is used as the data transport protocol, there are two types of commands that must be supported. The class commands are indicated in table 5.

Table 5 Class Commands

bRequest Field Value	Command	Meaning of Command
255 (H'FF)	Bulk-Only Mass Storage Reset	Resets the interface
254 (H'FE)	Get Max LUN	Checks the number of LUNs supported

When the Bulk-Only Mass Storage Reset command is received, the function resets all of the interfaces used in USB Mass Storage Class Bulk-Only Transport.

When the Get Max LUN command is received, the function returns the largest logical unit number that can be used. In the sample system used here, there is one logic unit, so a value of 0 will be returned to the host.

3.5 Sub-Class Code (SCSI Transparent Command Set)

The various commands must be processed in response to the sub-class commands in the CBW sent to the function by the host PC.

In this sample program, the eleven SCSI commands shown in table 6 are supported. If a command is not supported, the CSW will be used to inform the host PC that the command failed.

Table 6 Supported Commands

Operation Code	Command Name	Command Operation
H'00	TEST UNIT READY	Checks whether or not a medium can be used.
H'03	REQUEST SENSE	If an error occurred for the previous command, this tells the host what kind of error occurred.
H'12	INQUIRY	Tells the host the drive information.
H'1A	MODE SENSE (6)	Tells the host the drive status.
H'1B	STOP/START UNIT	Controls insertion/removal of media.
H'1E	PREVENT ALLOW MEDIUM REMOVAL	Inhibits/enables installing and removing media.
H'23	READ FORMAT CAPACITY	Tells the host the media format information.
H'25	READ CAPACITY	Tells the host the media sector information.
H'28	READ (10)	Reads the specified sector volume data from a specified sector.
H'2A	WRITE (10)	Writes the specified sector volume data to a specified sector.
H'2F	VERIFY (10)	Confirms whether or not the data in a medium can be accessed.

4. Development Environment

This chapter looks at the development environment used to develop this system. The devices (tools) listed below were used when developing the system.

- SH7216 CPU Board (part number: R0K572167) manufactured by Renesas Technology Corp.
- E10A-USB Emulator manufactured by Renesas Technology Corp.
- E10A PC (Windows® 2000, Windows® XP)
- USB host PC (Windows® 2000/Windows® XP/Windows® Vista or Mac OS9)
- USB cable
- High-performance Embedded Workshop 4 (hereafter called the HEW4) manufactured by Renesas Technology Corp.

4.1 Hardware Environment

Figure 3 shows device connections.

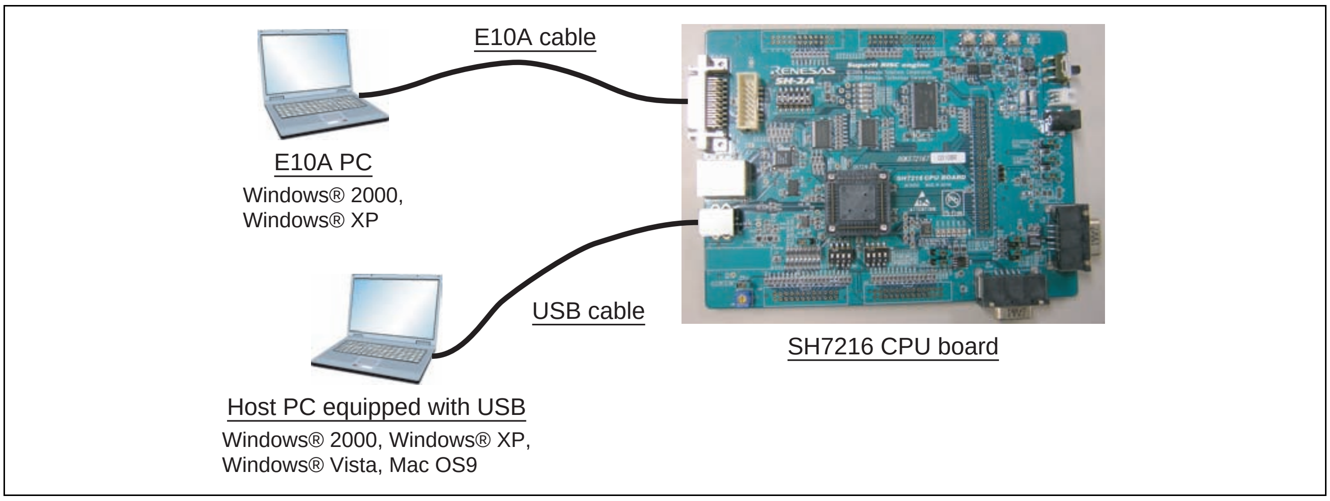

Figure 3 Device Connections

(1) SH7216 CPU Board

Because this system uses the on-chip ROM and SDRAM, the SH7216 CPU board must be operated in the MCU expansion mode 2 (both on-chip ROM and SDRAM enabled). Therefore, DIP switch SW1 on the SH7216 CPU board must be changed from the factory setting to the setting shown in table 7. Before turning on the power, ensure that the switches are set as follows. There is no need to change any other DIP switches.

Table 7 DIP Switch Settings

At Time of Shipment (Mode 6)	After Change (Mode 2)	DIP Switch Function
SW1-1 (FWE) OFF	SW1-1 (FWE) ON	On-chip flash memory write/erase protection
SW1-2 (MD1) OFF	SW1-2 (MD1) OFF	MD1 pin state
SW1-3 (MD0) ON	SW1-3 (MD0) ON	MD0 pin state

(2) USB host PC

A PC with Windows® 2000/Windows® XP/Windows® Vista or Mac OS9 installed, and with a USB port, is used as the USB host. This system uses USB Mass Storage Class (Bulk-Only Transport) device drivers installed as a standard part of the Windows® XP system, and so there is no need to install new drivers.

(3) E10A PC

A PC with Windows® 2000/Windows® XP installed, and with a USB port, is used as the E10A PC. Connect the E10A-USB emulator to the USB connector on the E10A-USB PC, and then connect the E10A-USB emulator to the CPU board with the cable. After connection, start the HEW4 and perform emulation.

4.2 Software Environment

Compile, link, and debug the source code with HEW4. To start HEW4, double-click "MSC.hws" in this folder.

4.2.1 Sample Program

Files required for the sample program are all stored in the MSC folder. When this entire folder with its contents is moved to a PC on which HEW4 have been installed, the sample program can be used immediately. Files included in the folder are indicated in figure 4 below.

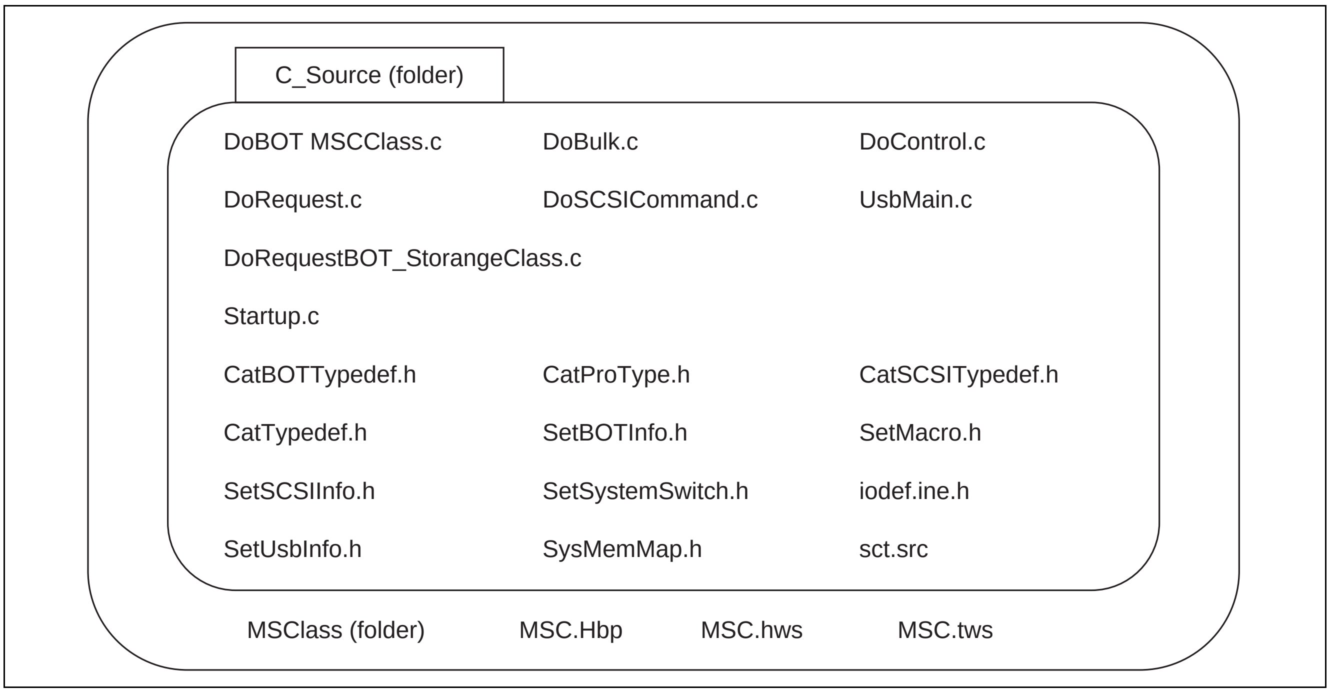

Figure 4 Files Included in the Folder

4.2.2 Compiling and Linking

Compile the source code with HEW4.

4.3 Loading and Executing the Program

Allocation of programs to respective areas is specified by "lnkSet.sub" in the sh7216_usb_msc folder. The allocation result is output to "MSClass.map" in the sh7216_usb_msc \MSClass\Debug folder. When changing the layout of the programs, change "lnkSet.sub".

4.3.1 Loading the Program

In order to load the sample program into the SDRAM of the SH7216 CPU Board, the following procedure is used.

- Connect the E10A-USB to the E10A-USB PC with HEW4 installed.
- Connect the E10A-USB to the SH7216 CPU board with the user cable.
- Turn on the power to the SH7216 CPU Board.
- Execute "MSC.hws" in the sh7216_usb_msc folder.
- Select "Debug > Connection."
- You are asked to select a mode of the emulator. Select "SH7216(R5F72167A)" or "E10A-USB Emulator."
- Press the reset switch on the SH7216 CPU board, and then click on the OK button.
- You are asked to enter an operating frequency. Enter the frequency (12.50 MHz) of the mounted crystal oscillator.
- You are asked to enter an ID code. Enter E10A.
- Select "Debug > Download > All Download Modules" to download the sample program.

4.3.2 Executing the Program

Select "Debug > Execute after Reset" to execute the sample program.

4.4 Using the RAM Disk

The following describes an example in which Windows® XP is used.

After the program has been run, the Series B connector of the USB cable is inserted into the SH7216 CPU Board, and the Series A connector on the opposite side is connected to the USB host PC.

After the emulation used for control transfer and bulk transport has ended, USB Large-Size Storage Device is displayed under USB Controller in the Device Manager, and RENESAS EX RAM Disk USB Device is displayed under Disk Drive. As a result, the host PC recognizes the SH7216 CPU Board as the storage device, and the local disk is mounted in the microcomputer.

Next, the local disk needs to be formatted.

Select Local Disk and click with the right button of the mouse to display a floating menu. Select Format. A format selection window for the drive is displayed. Enter the necessary format settings. Check to make sure FAT has been selected for the file system, and click on the Start button.

A format confirmation window is displayed. Click on the OK button.

When the formatting has been completed, a message window is displayed. Click on the OK button.

The screen returns to the drive format selection window. Click on the Close button to exit the procedure.

The SH7216 CPU Board can now be used as the RAM disk for USB connection.

4.5 Changing RAM Disk Settings

This section describes how to change the settings of the RAM disk used in this sample program.

4.5.1 Removable/Hard Disk

The RAM disk is used as a removable disk in this sample program.

The RAM disk can be used as a hard disk by commenting out "#define REMOVABLE_DISK" in "SetSystemSwitch.h" and enabling comment "#undef REMOVABLE_DISK."

4.5.2 Changing the RAM Disk Capacity

This sample program uses 16-Mbyte capacity of the SDRAM as a RAM disk. To change the RAM disk capacity, change the content of "SysMemMap.h." Specify the entire byte count *1 to be used as a RAM disk by DISK_ALL_BYTE. Then specify the start address and end address of the area to be used as a RAM disk by RAM_DISK _S and RAM_ DISK _E *2, respectively.

Notes: 1. Specify a value of 1.5 Mbytes or more. The capacity viewed from the PC is slightly reduced because the FAT information and other information use the capacity. This sample program configures the FAT information with FAT12 (up to approx. 16 Mbytes) and FAT16 (up to approx. 2 Gbytes). Other FAT system information need to be prepared by the user.
2. The area specified by RAM_ DISK _S and RAM_ DISK _E must be equal to or larger than the size specified by DISK_ALL_BYTE.

5. Overview of the Sample Program

In this section, features of the sample program and its structure are explained. This sample program runs on the SH7216 CPU Board, which works as a RAM disk, and initiates USB transfers by means of interrupts from the USB function module. Of the interrupts from modules in the SH7216, there are six interrupts related to the USB function module: USI0, USI1, USBRXI0, USBTXI0, USBRXI1 and USBTXI1, but in this sample program, USI0 and USI1 are used.

5.1 State Transition Diagram

Figure 5 shows a state transition diagram for this sample program. In this sample program, as shown in figure 5, there are transitions between three states.

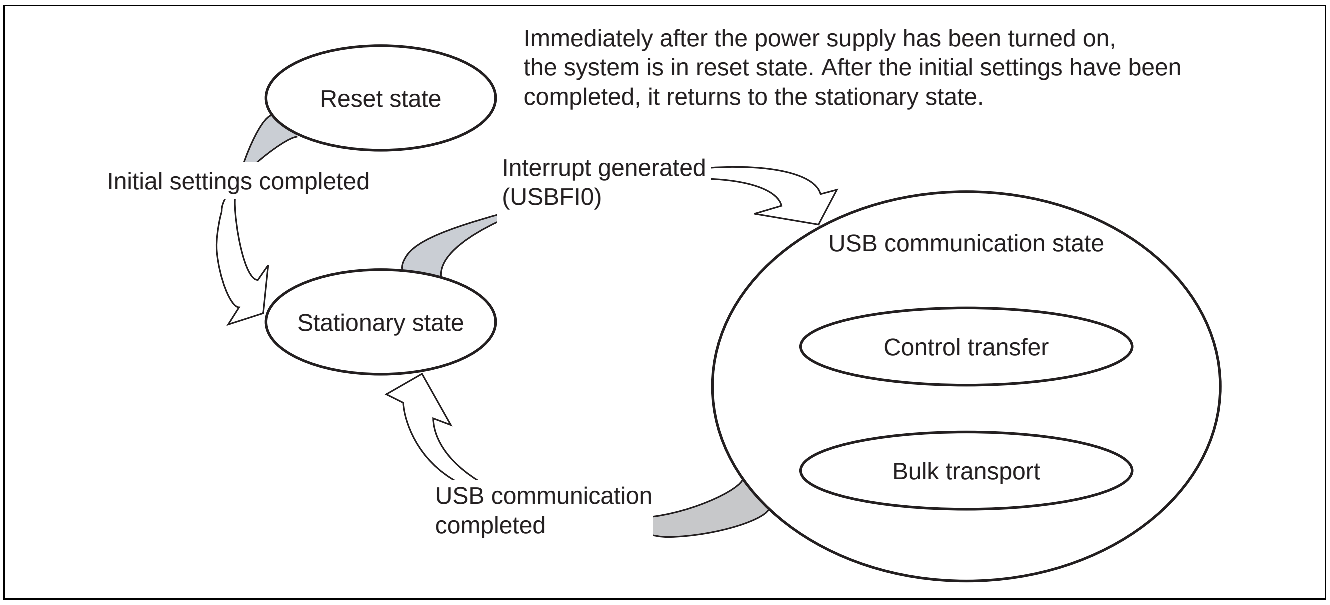
Immediately after the power supply has been turned on, the system is in reset state. After the initial settings have been completed, it returns to the stationary state.
Reset state
Initial settings completed
Interrupt generated (USBFI0)
USB communication state
Stationary state
Control transfer
Bulk transport
USB communication completed

Figure 5 State Transition Diagram

- Reset state
  Upon power-on reset and manual reset, this state is entered. In the reset state, the SH7216 mainly performs initial settings.
- Stationary state
  When initial settings are completed, a stationary state is entered in the main loop.
- USB communication state
  In the stationary state, when an interrupt from the USB module occurs, this state is entered. In the USB communication state, data transfer is performed by a transfer method according to the type of interrupt. The interrupts used in this sample program are indicated by interrupt flag register 0, 1, 2, 3 and 4 (USBIFR0, 1, 2, 3 and 4). When an interrupt factor occurs, the corresponding bits in USBIFR0, 1, 2, 3 and 4 are set.

5.2 USB Communication State

The USB communication state can be further divided into three states according to the transfer type (see figure 6). When an interrupt occurs, first there is a transition to the USB communication state, and then there is further branching to a transfer state according to the interrupt type.

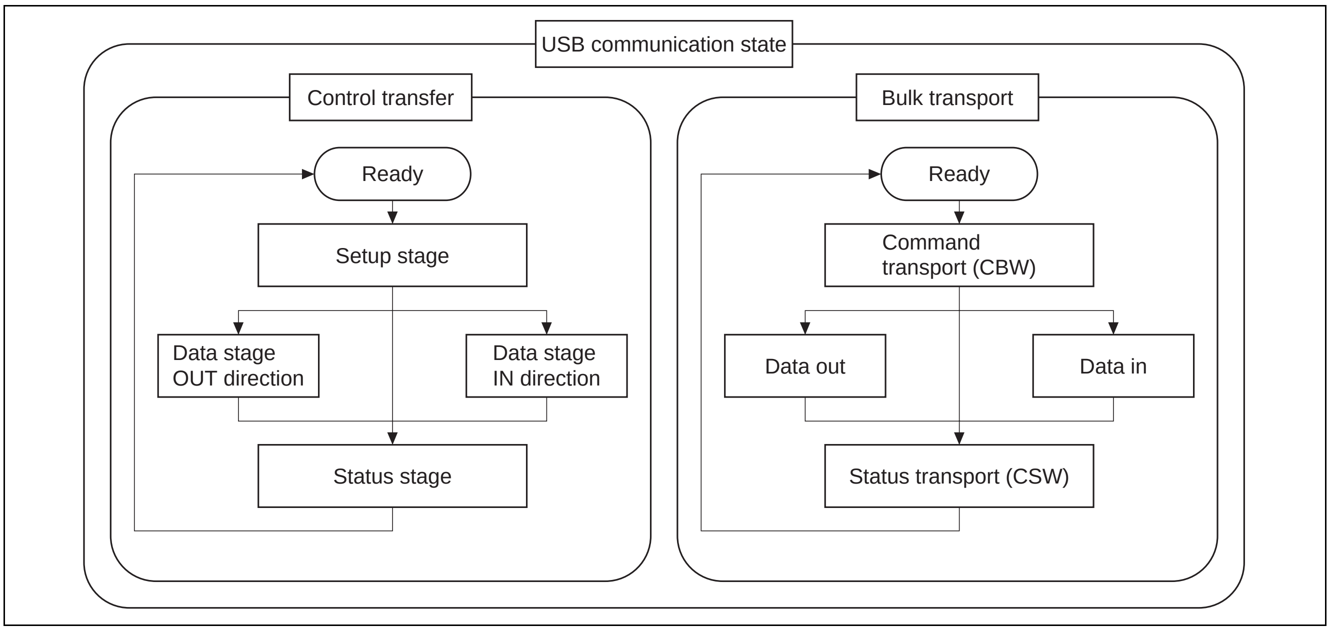

Figure 6 USB Communication State

5.2.1 Control Transfer

Control transfer is used mainly for functions such as obtaining device information and specifying device operating states. For this reason, when the function is connected to the host PC, control transfer is the first transport to be carried out.

Transport processing for control transfer is carried out in a series of two or three stages. These stages are a setup stage, a data stage, and a status stage.

5.2.2 Bulk Transport

Bulk transport has no time limitations, so it is used to send large volumes of data with no errors. The data transport speed is not guaranteed, but the data contents are guaranteed. With USB Mass Storage Class (Bulk-Only Transport), bulk transport is used to transfer storage data between the host PC and the function.

Transport processing for USB Mass Storage Class (Bulk-Only Transport) is carried out in a series of two or three stages. These stages are command transport (CBW), data transport, and status transport (CSW).

5.3 File Structure

This sample program consists of nine source files and eleven header files. The overall file structure is shown in table 8. Each function is arranged in one file by transfer method or function type.

Table 8 File Structure

File Name	Principle Role
Startup.c	USB function default settings
UsbMain.c	Judging the causes of interrupts Sending and receiving packets
DoRequest.c	Processing setup commands issued by the host
DoRequestBOT_StorageClass.c	Processing USB Mass Storage Class (Bulk-Only Transport) class commands
DoControl.c	Executing control transfer
DoBulk.c	Executing bulk transport
DoBOTMSClass.c	Executing USB Mass Storage Class (Bulk-Only Transport) transport
DoSCSICommand.c	Analyzing and processing SCSI commands
sct.src	Transferring initial values of variables from the ROM to the RAM
CatBOTTypedef.h	Defining structures used for Bulk-Only Transport
CatProType.h	Prototype declarations
CatSCSITypedef.h	Defining structures used for SCSI, and defining macro to configure FAT information
CatTypedef.h	Defining the basic structures used in USB firmware
SetBOTInfo.h	Default settings of variables needed to support Bulk-Only Transport
SetMacro.h	Defining macros
SetSCSIInfo.h	Default settings of variables needed to support SCSI commands
SetSystemSwitch.h	System operation settings
SetUsbInfo.h	Default settings of variables used in USB firmware
SysMemMap.h	Defining memory map addresses
iodefine.h	Defining SH7216 registers

5.4 Purposes of Functions

Table 9 to 16 shows functions contained in each file and their purposes.

- Startup.c
  When a power-on reset or manual reset is carried out, the SetPowerOnSection of the Startup.c file is called. At this point, the SH7216 default settings are entered and the RAM area used for bulk transport is cleared.

Table 9 Startup.c

File in which Stored	Function Name	Purpose
Startup.c	SetPowerOnSection	Initializes module and memory, and shifts to the main loop
	InitSDRAM	Makes initial settings for the SDRAM mounted on the SH7216 CPU board
	_INITSCT	Copies variables that have default settings to the RAM work area
	InitMemory	Clears the RAM area used in bulk communication
	InitSystem	Pull-up control of the USB bus
	Set_EPInfoR	Writes endpoint information

- UsbMain.c
  In UsbMain.c, interrupt factors are discriminated by the USB interrupt flag register, and functions are called according to the interrupt type. Also, packets are sent and received between the host controller and function modules.

Table 10 UsbMain.c

File in which Stored	Function Name	Purpose
UsbMain.c	BranchOfInt0	Performs a bus reset, determines endpoint0 interrupt source, and calls a function corresponding to the interrupt
	BranchOfInt1	Determines endpoint1 to endpoint9 interrupt sources, and calls a function corresponding to each interrupt
	GetPacket	Writes data transferred from the host controller to RAM
	GetPacket4	Writes data transferred from the host controller to RAM in longwords (ring buffer supported) (not used by this sample program)
	GetPacket4S	Writes data transferred from the host controller to RAM in longwords (ring buffer not supported, high-speed version)
	PutPacket	Writes data for transfer to the host controller to the USB module
	PutPacket4	Writes data for transfer to the host controller to the USB module in longwords (ring buffer supported) (not used by this sample program)
	PutPacket4S	Writes data for transfer to the host controller to the USB module in longwords (ring buffer not supported, high-speed version)
	SetControlOutContents	Overwrites data with that sent from the host
	SetUsbModule	Makes USB module initial settings
	ActBusReset	Clears FIFO on receiving bus reset
	ActBusVcc	Generates a USB cable connection interrupt (not used by this sample program)
	ConvRealn	Reads data of a specified byte length from a specified address
	ConvReflexn	Reads data of a specified byte length from specified addresses, in reverse order

- DoRequest.c
  During control transfer, commands sent from the host controller are decoded and processed. In this sample program, a vendor ID of H'045B (vendor: Renesas) is used. When the customer develops a product, the customer should obtain a vendor ID at the USB Implementers' Forum. Because vendor commands are not used, DecVenderCommands does not perform any action. In order to use a vendor command, the customer should develop a program.

Table 11 DoRequest.c

File in which Stored	Function Name	Purpose
DoRequest.c	DecStandardCommands	Decodes command issued by host controller, and processes standard commands
	DecVenderCommands	Processes vendor commands

- DoRequestBOT_StorageClass.c
  This function carries out processing according to the USB Mass Storage Class (Bulk-Only Transport) commands (Bulk-Only Mass Storage Reset and Get Max LUN).
  The Bulk-Only Mass Storage Reset command resets all of the interfaces used in Bulk-Only Transport.
  The Get Max LUN command returns the largest logical unit number used by peripheral devices. In this sample program, there is one logical unit, so a value of 0 is returned to the host.

Table 12 DoRequestBOT_StorageClass.c

File in which Stored	Function Name	Purpose
DoRequestBOT_StorageClass.c	DecBOTClassCommands	Processes USB Mass Storage Class (Bulk-Only Transport) commands

- DoControl.c
  When control transfer interrupt SETUP TS is generated, ActControl obtains the command, and decoding is carried out by DecStandardCommands to determine the transfer direction. Next, when control transfer interrupt EP0o TS, EP0i TR, or EP0i TS is generated, ActControlInOut calls either ActControlIn or ActControlOut depending on the transfer direction, and the data stage and status stage are carried out by the called function.

Table 13 DoControl.c

File in which Stored	Function Name	Purpose
DoControl.c	ActControl	Controls the setup stage of control transfer
	ActControlIn	Controls the data stage and status stage of control IN transport (transport in which the data stage is in the IN direction)
	ActControlOut	Controls the data stage and status stage of control OUT transport (transport in which the data stage is in the OUT direction)
	ActControlInOut	Sorts the data stage and status stage of control transfers and direct them to ActControlIn and ActControlOut.

- DoBulk.c
  These functions carry out processing involving bulk transport. ActBulkInReady is not used in USB Mass Storage Class (Bulk-Only Transport).

Table 14 DoBulk.c

File in which Stored	Function Name	Purpose
DoBulk.c	ActBulkOut	Performs bulk-out transfer
	ActBulkIn	Performs bulk-in transfer
	ActBulkInReady	Performs preparations for bulk-in transfer

- DoBOTMSClass.c
  With DoBOTMSClass.c, control of the two or three stages of the USB Mass Storage Class (Bulk-Only Transport) is carried out, and operation is carried out in accordance with the specifications.

Table 15 DoBOTMSClass.c

File in which Stored	Function Name	Purpose
DoBOTMSClass.c	ActBulkOnly	Divides Bulk-Only Transport into separate stages
	ActBulkOnlyCommand	Controls CBW for Bulk-Only Transport
	ActBulkOnlyIn	Controls data transport and status transport (when the data stage is in the IN direction)
	ActBulkOnlyOut	Controls data transport and status transport (when the data stage is in the OUT direction)

- DoSCSICommand.c
  The DoSCSICommand.c function is used to analyze SCSI commands sent from the host PC and prepare for the next data transport or status transport.

Table 16 DoSCSICommand.c

File in which Stored	Function Name	Purpose
DoSCSICommand.c	DecBotCmd	Processes SCSI commands sent from the host using Bulk-Only Transport
	SetBotCmdErr	Processes SCSI command errors

5.5 RAM Disk

In the sample program provided here, the SD-RAM in the SH7216 CPU Board is selected as the disk device, and the host PC is notified that the SH7216 CPU Board (function) is the disk.

As shown in figure 7, the disk device of the function has a master boot block and a partition boot block. When the system is booted, an initialization routine is used to write the master boot block and the partition boot block to the RAM disk area on the SDRAM.

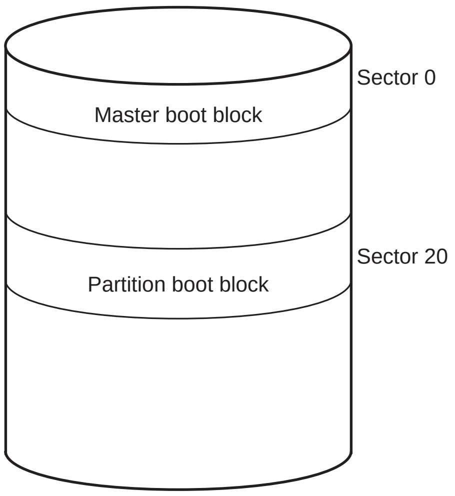

Figure 7 Disk Construction

SCSI commands are used to allow function access from the host PC (saving and loading data). In order to work with SCSI commands, the user needs to understand the construction shown in figure 7 and then write the operation.

5.6 Operation of SCSI Commands That Are Supported

Table 17 shows the SCSI commands that are supported by the sample program.

Table 17 SCSI Command Operations

Command Name	Transport Name	Operation Content
INQUIRY	CBW	This decodes a command and recognizes it as an INQUIRY command. It then prepares to send the INQUIRY information (96 bytes) stored in the ROM.
	Data	This sends the INQUIRY information to the host PC using bulk-in transport.
	CSW	This sends the results of executing a command to the PC. If the data being sent is 96 bytes or less, the transmission will end successfully.
READ CAPACITY	CBW	This decodes a command and recognizes it as a READ CAPACITY command. It then prepares to send the 8-byte READ CAPACITY information (consisting of byte count per sector and total sector count) of the disk on the SDRAM. If media access is disabled (LSB of unit_state[0] is 1), the function treats this as no data transfer and follows (4) in section 5.7, Processing If an Error Occurs. It also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	This sends the READ CAPACITY information to the host PC using bulk-in transport. If media access is disabled, it returns data as much as data requested by the host (H'00).
	CSW	This sends the results of the command execution to the host PC. If media access is disabled, it returns a command fail (CSW status H'01).
READ (10)	CBW	This decodes the command and recognizes it as the READ (10) command. It then prepares to send the data for a specified read sector volume from the Disk device open on the SDRAM. If media access is disabled (LSB of unit_state[0] is 1), the function treats this as no data transfer and follows (4) in section 5.7, Processing If an Error Occurs. It also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	This sends the data from the read sectors to the host PC using bulk-in transport. If media access is disabled, it returns data as much as data requested by the host (H'00).
	CSW	This sends the results of executing the READ (10) command to the host PC. If media access is disabled, it returns a command fail (CSW status H'01).

Command Name	Transport Name	Operation Content
WRITE (10)	CBW	This decodes the command and recognizes it as the WRITE (10) command. It then prepares to receive the data of the specified sector volume from the specified write sector in the Disk device open on the SDRAM. If media access is disabled (LSB of unit_state[0] is 1), the function treats this as no data transfer and follows (9) in section 5.7, Processing If an Error Occurs. It also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	This receives the write sector data from the host PC using bulk-out transport. If media access is disabled, it reads and discards data from the host.
	CSW	This notifies the host PC that the operation has been completed successfully. If media access is disabled, it returns a command fail (CSW status H'01).
REQUEST SENSE	CBW	This decodes the command and recognizes it as the REQUEST SENSE command. It then prepares to send the returned value (the results of executing the previous SCSI command).
	Data	This sends the returned value to the host PC using bulk-in transport.
	CSW	This sends the results of the command execution to the host PC. The transmission is completed successfully as long as the data consists of 18 bytes or less.
PREVENT ALLOW MEDIUM REMOVAL	CBW	This decodes the command and recognizes it as the PREVENT ALLOW MEDIUM REMOVAL command. It then prepares to notify the host PC that the operation has been successfully completed. If media access is disabled (LSB of unit_state[0] is 1), it sets the command to "fail" and also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	Data transport does not exist for this command.
	CSW	This notifies the host PC that the operation has been completed successfully. If media access is disabled, it returns a command fail (CSW status H'01).
TEST UNIT READY	CBW	This decodes the command and recognizes it as the TEST UNIT READY command. It then prepares to notify the host PC that the operation has been successfully completed. If media access is disabled (LSB of unit_state[0] is 1), it sets the command to "fail" and also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	Data transport does not exist for this command.
	CSW	This notifies the host PC that the operation has been completed successfully. If media access is disabled, it returns a command fail (CSW status H'01).

Command Name	Transport Name	Operation Content
VERIFY (10)	CBW	This decodes the command and recognizes it as the VERIFY (10) command. It then prepares to notify the host PC that the operation has been successfully completed. If media access is disabled (LSB of unit_state[0] is 1), it sets the command to "fail" and also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	Data transport does not exist for this command.
	CSW	This notifies the host PC that the operation has been completed successfully. If media access is disabled, it returns a command fail (CSW status H'01).
STOP/START UNIT	CBW	This decodes a command and recognizes it as a STOP/START UNIT command. It then sets the LSB of global variable unit_state[0] to 1 when the command specifies media ejection or stop, or clears the LSB of global variable unit_state[0] to 0 in other cases. When the user wants recovery from the inaccessible state, clear the LSB of unit_state[0] to 0.
	Data	Data transport does not exist for this command.
	CSW	This notifies the host PC that the operation has been completed successfully.
MODE SENSE (6)	CBW	This decodes a command and recognizes it as a MODE SENSE (6) command. It then prepares to send the requested MODE SENSE information.
	Data	This sends the MODE SENSE information to the host PC using bulk-in transport.
	CSW	This sends the results of the command execution to the host PC.
READ FORMAT CAPACITY	CBW	This decodes a command and recognizes it as a READ FORMAT CAPACITY command. Regarding that the disk on the SDRAM has been formatted, it then prepares to send the 20-byte READ FORMAT CAPACITY information (consisting of byte count per sector and total sector count of the disk). If media access is disabled (LSB of unit_state[0] is 1), the function treats this as no data transfer and follows (4) in section 5.7, Processing If an Error Occurs. It also sets a value to be returned in REQUEST SENSE for NOT READY (not prepared).
	Data	This sends the READ FORMAT CAPACITY information to the host PC using bulk-in transport. If media access is disabled, it returns data as much as data requested by the host (H'00).
	CSW	This sends the results of the command execution to the host PC. If media access is disabled, it returns a command fail (CSW status H'01).

Command Name	Transport Name	Operation Content
Commands that are not supported	CBW	This decodes the command and, if it is an unsupported command, specifies INVALID FIELD IN CDB for the returned value of the REQUEST SENSE command. It then prepares to transport the data.
	Data	If the host PC has requested data using bulk-in transport, this sends the same volume of data (H'00) as that requested by the host PC. If the host PC has sent data using bulk-out transport, the number of bytes received are counted. If there is no data transport, no operation is carried out.
	CSW	This sends a command fail (CSW status H'01) to the host PC.

5.7 Processing If an Error Occurs

The errors that may occur during a USB Mass Storage Class (Bulk-Only Transport) transmission between the host PC and function, and how the function operates when an error occurs are described below.

The Bulk-Only Transport standard defines the following two types of errors:

— Invalid CBW
— Operation expected by the host PC and operation planned by the function (operation specified by the SCSI command) do not match (10 cases)

The Bulk-Only Transport standard does not cover any other states.

There are 13 states for data transfer between the host PC and a function as shown in tables 18 and 19. Cases (1), (6) and (12) are normal transport states.

Table 18 Data Transport States between Host PC and Function

		What the Host PC Expects		
		No Data Transport	**Data Reception from Function**	**Data Send to Function**
What the function plans	No data transport	(1) Hn = Dn	(4) Hi > Dn	(9) Ho > Dn
	Data send to host PC	(2) Hn < Di	(5) Hi > Di	(10) Ho < > Di
			(6) Hi = Di	
			(7) Hi < Di	
	Data reception from host PC	(3) Hn < Do	(8) Hi < > Do	(11) Ho > Do
				(12) Ho = Do
				(13) Ho < Do

Table 19 Explanation of Data Transport States between Host PC and Function

Case No.	Relation between Host PC and Function
(1)	The host PC expects no data transport and the function plans no data transport.
(2)	The host PC expects no data transport but the function plans to send data to the host PC.
(3)	The host PC expects no data transport but the function plans to receive data from the host PC.
(4)	The host PC expects to receive data from the function but the function plans no data transport to the host PC.
(5)	The amount of data the function sends to the host PC is less than the amount of data the host PC expected to receive from the function.
(6)	The amount of data the function sends to the host PC is equal to the amount of data the host PC expected to receive from the function.
(7)	The amount of data the function sends to the host PC is greater than the amount of data the host PC expected to receive from the function.
(8)	The host PC expects to receive data from the function but the function plans to receive data from the host PC.
(9)	The host PC expects to send data to the function but the function plans no data transport to the host PC.
(10)	The host PC expects to send data to the function but the function plans to send data to the host PC.
(11)	The amount of data the function receives from the host PC is less than the amount of data the host PC expected to send to the function.
(12)	The amount of data the function receives from the host PC is equal to the amount of data the host PC expected to the function.
(13)	The amount of data the function receives from the host PC is greater than the amount of data the host PC expected to send to the function.

Table 20 shows sample error conditions that may be generated.

Table 20 Sample Error Conditions

Case No.	Error Conditions
(2)	When a READ command is issued from the host PC, the amount of data to be transported in the USB data transport is 0 while the amount of data specified by the SCSI command is a value other than 0.
(3)	When a WRITE command is issued from the host PC, the amount of data to be transported in the USB data transport is 0 while the amount of data specified by the SCSI command is a value other than 0.
(4)	When a READ command is issued from the host PC, the amount of data to be transported in the USB data transport is 0 while the amount of data specified by the SCSI command is 0.
(5)	When a READ command is issued from the host PC, the amount of data specified by the SCSI command is less than the amount of data to be transported in the USB data transport.
(7)	When a READ command is issued from the host PC, the amount of data specified by the SCSI command is greater than the amount of data to be transported in the USB data transport.
(8)	Even though a WRITE command has been issued from the host PC, the host PC requests for data in the USB data transport.
(9)	When a WRITE command is issued from the host PC, the amount of data to be transported in the USB data transport is a value other than 0 while the amount of data specified by the SCSI command is 0.
(10)	Even though a READ command has been issued from the host PC, the host PC sends data in the USB data transport.
(11)	When a WRITE command is issued from the host PC, the amount of data specified by the SCSI command is less than the amount of data to be transported in the USB data transport.
(13)	When a WRITE command is issued from the host PC, the amount of data specified by the SCSI command is greater than the amount of data to be transported in the USB data transport.

Table 21 shows how a function operates when each error condition occurs.

Table 21 Function Operation for Each Error Condition

Case No.	Relation between Host PC and Function
(2), (3)	• Set H'02 as the CSW status.
(4), (5)	• The function adds data to become equal to the data length set in dCBWDataTransferLength and then sends data to the host PC. • Set the amount of data added in the data transport in dCBWDataResidue of CSW. • Set H'00 as the CSW status.
(7), (8)	• The function sends data to the host PC up to the data length set in dCBWDataTransferLength. • Set H'02 as the CSW status.
(9), (11)	• The function receives data from the host PC up to the data length set in dCBWDataTransferLength. • Set the difference between the amount of data received in the data transport and the amount of data processed by the function in dCBWDataResidue of CSW. • Set H'01 as the CSW status.
(10), (13)	• The function receives data from the host PC up to the data length set in dCBWDataTransferLength. • Set H'02 as the CSW status.

Figures 8 to 10 show the processing when a data transport error occurs.

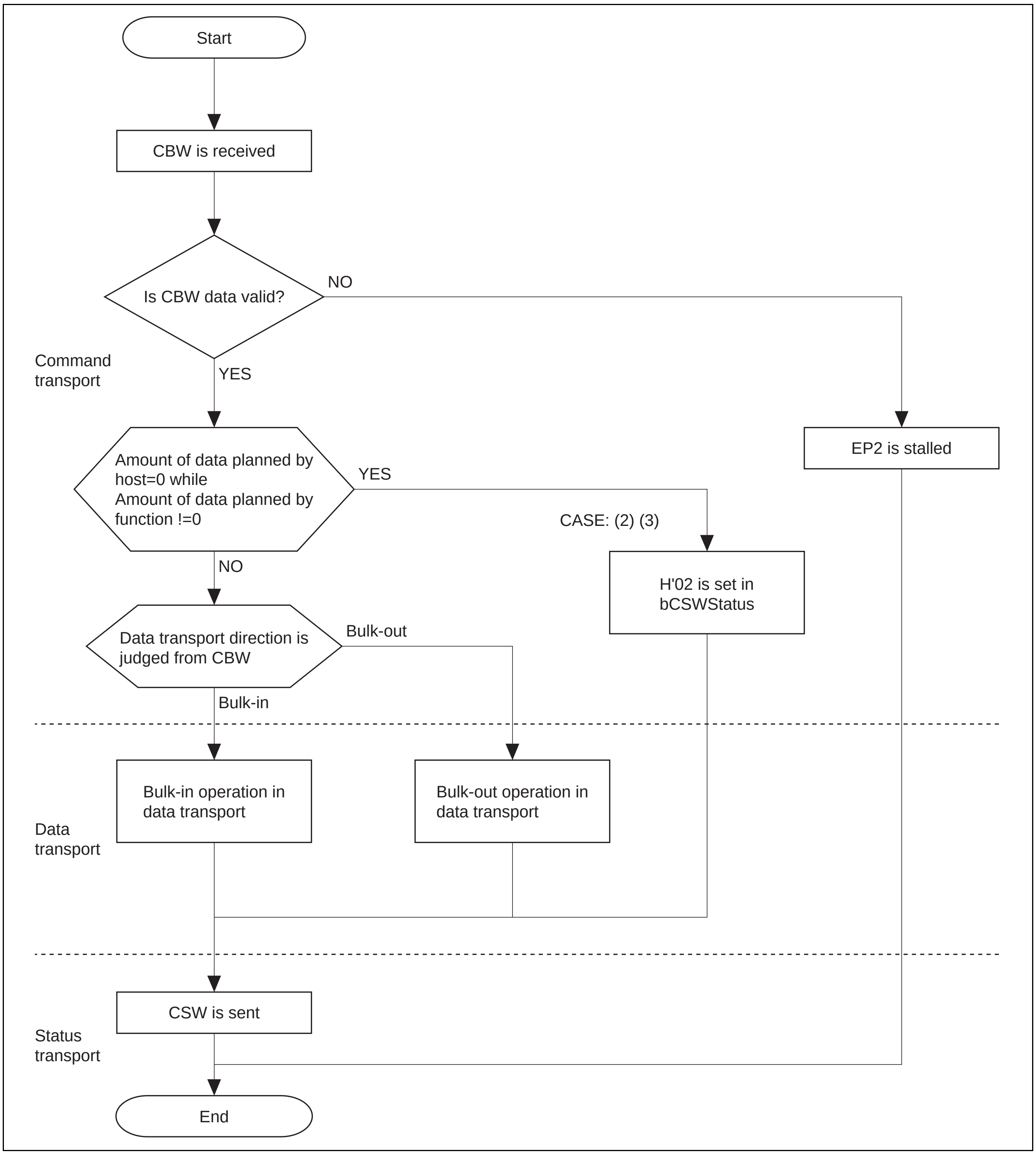
Start
CBW is received
Is CBW data valid?
NO
YES
Command transport
EP2 is stalled
Amount of data planned by host=0 while Amount of data planned by function !=0
YES
CASE: (2) (3)
H'02 is set in bCSWStatus
NO
Data transport direction is judged from CBW
Bulk-out
Bulk-in
Bulk-in operation in data transport
Bulk-out operation in data transport
Data transport
CSW is sent
Status transport
End

Figure 8 Error Processing Flow in Data Transport (1)

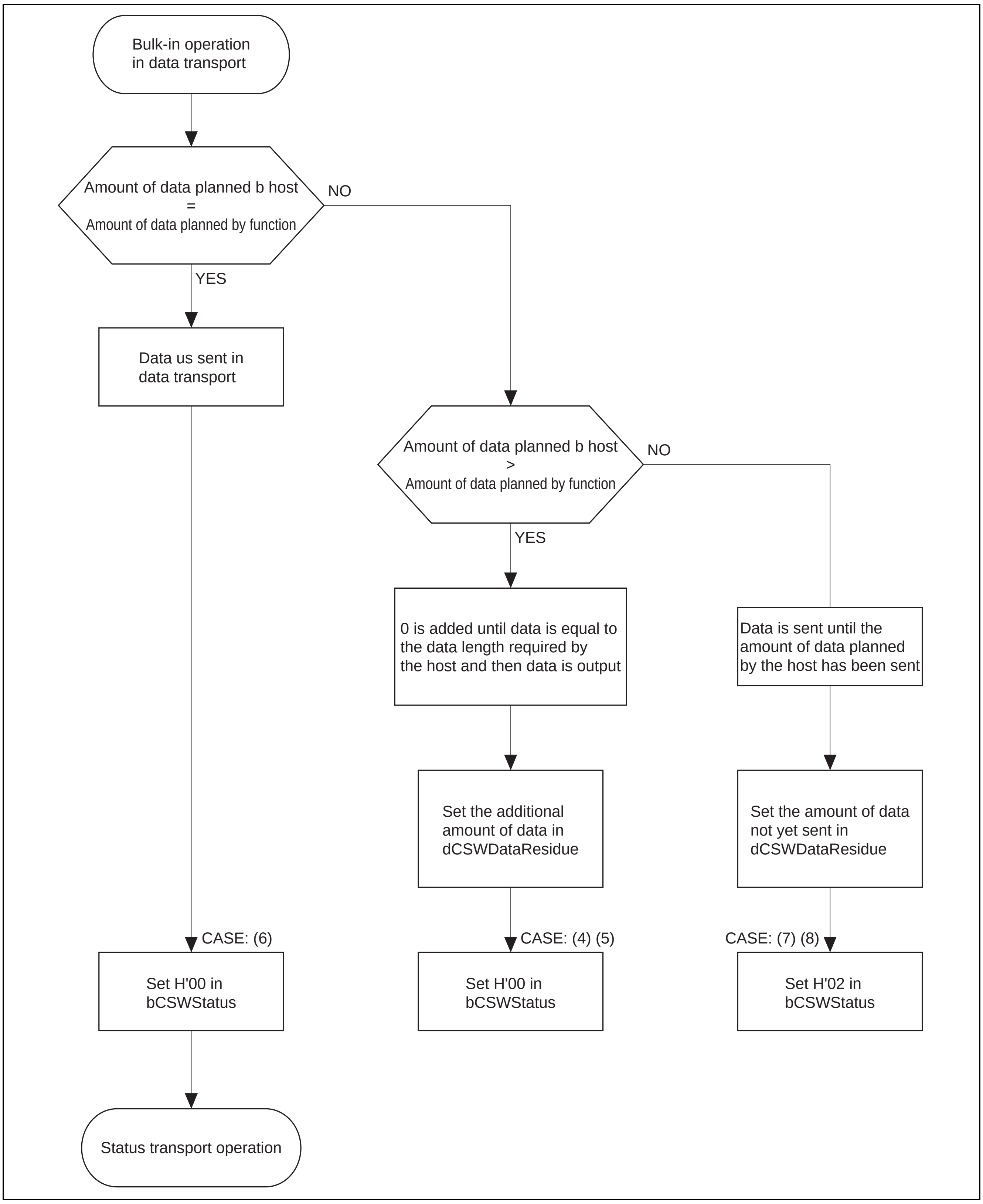
Bulk-in operation in data transport
Amount of data planned b host = Amount of data planned by function
NO
YES
Data us sent in data transport
Amount of data planned b host > Amount of data planned by function
NO
YES
0 is added until data is equal to the data length required by the host and then data is output
Data is sent until the amount of data planned by the host has been sent
Set the additional amount of data in dCSWDataResidue
Set the amount of data not yet sent in dCSWDataResidue
CASE: (6)
CASE: (4) (5)
CASE: (7) (8)
Set H'00 in bCSWStatus
Set H'00 in bCSWStatus
Set H'02 in bCSWStatus
Status transport operation

Figure 9 Error Processing Flow in Data Transport (2)

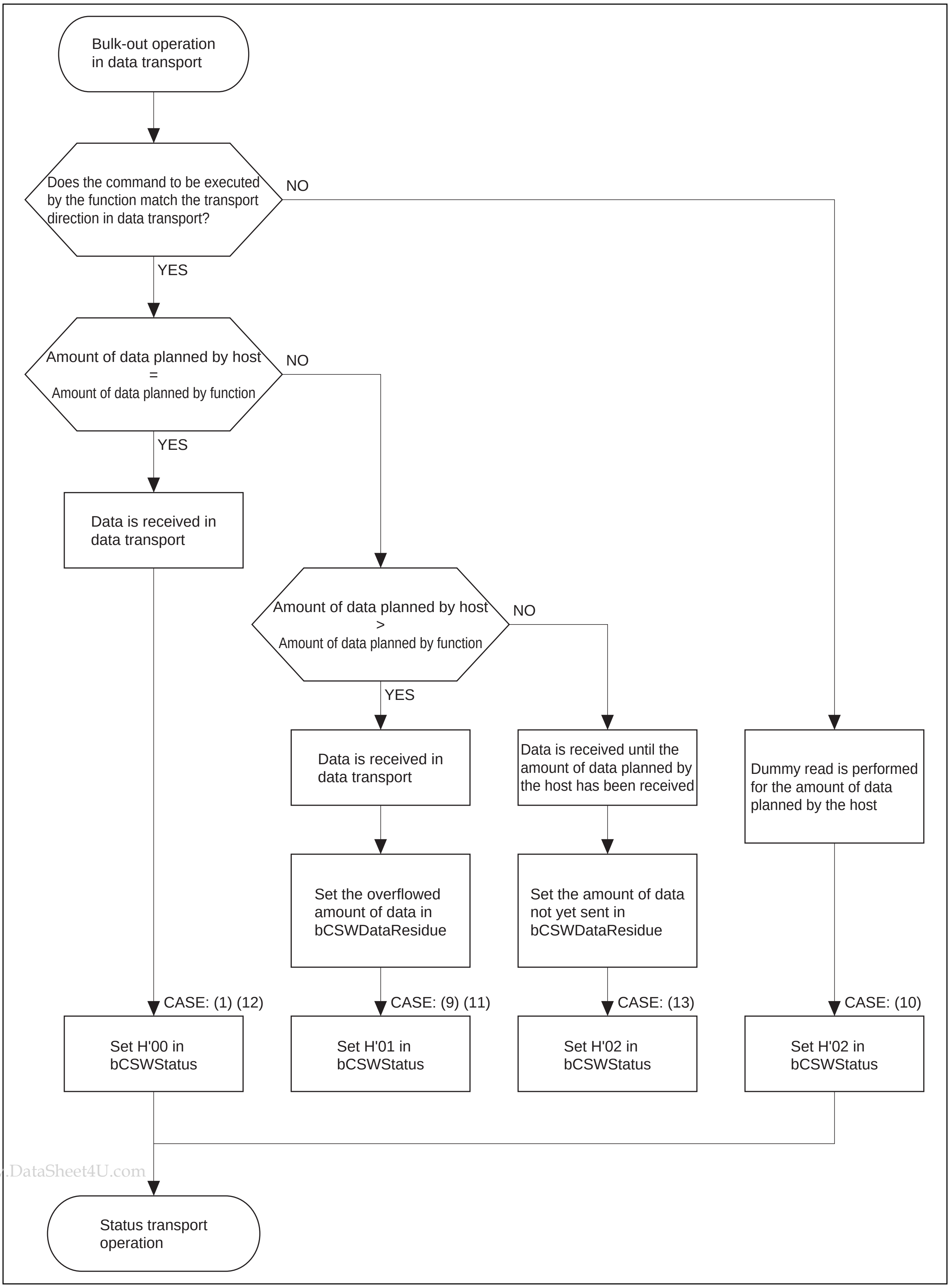
Bulk-out operation in data transport
Does the command to be executed by the function match the transport direction in data transport?
NO
YES
Amount of data planned by host = Amount of data planned by function
NO
YES
Data is received in data transport
Amount of data planned by host > Amount of data planned by function
NO
YES
Data is received in data transport
Data is received until the amount of data planned by the host has been received
Dummy read is performed for the amount of data planned by the host
Set the overflowed amount of data in bCSWDataResidue
Set the amount of data not yet sent in bCSWDataResidue
CASE: (1) (12)
CASE: (9) (11)
CASE: (13)
CASE: (10)
Set H'00 in bCSWStatus
Set H'01 in bCSWStatus
Set H'02 in bCSWStatus
Set H'02 in bCSWStatus
Status transport operation

Figure 10 Error Processing Flow in Data Transport (3)

When a USB Mass Storage Class (Bulk-Only Transport) transmission is carried out, transport of the CBW initiates a series of data transfers, and when the CSW is transported to the host PC, a series of data transfers is processed. This status contains two items: dCSWStatus that indicates the transport result, and dCSWDataResidue that indicates the number of error bytes.

In this sample program, the following two fields are used to create these two items.

— dCBWDataTransferLength field of CBW packet
— dCSWDataTransferResidue field of CSW packet

The dCBWDataTransferLength field of the CBW packet is used as the variable in which the number of data bytes the host PC specifies to be handled in the data transport is entered.

The dCSWDataTrasferResidue field of the CSW packet is used as the variable in which the number of data bytes the function handles in the data transport is entered.

When the CBW transport has been completed, the number of data bytes planned to be handled in the data transport by the host PC and the function are stored in the dCBWDataTransferLength and dCSWDataTransferResidue fields, respectively.

Data is transferred in the data transport according to the flowcharts.

If data transport between the host PC and function has been processed without errors, the values in the dCBWDataTransferLength and dCSWDataTransferResidue fields are both subtracted by the number of bytes that have been transferred for every data transfer in the data transport. For other cases, the difference between the number of data bytes the host PC requires to be handled in the data transport and the number of data bytes the function has handled in the data transport is stored in the dCSWDataTransferResidue field of the CSW packet, and operation then moves to the status transport.

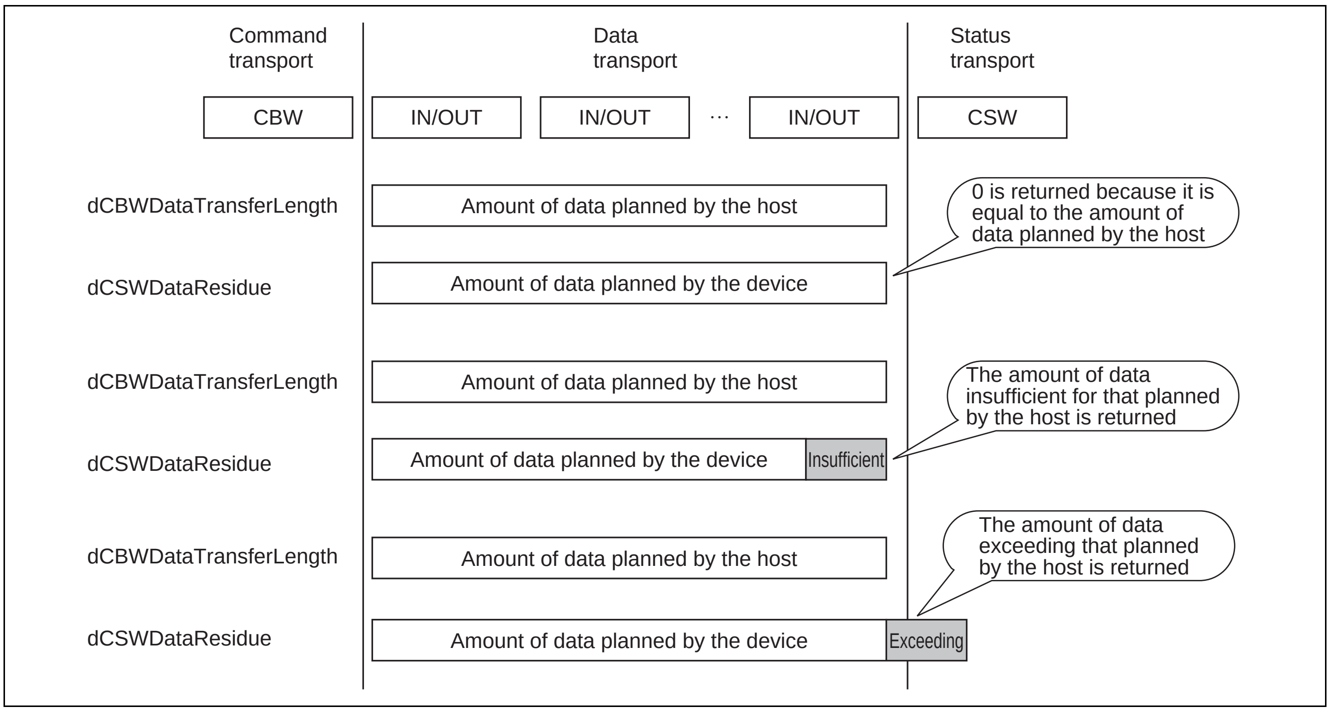
Command transport
Data transport
Status transport
CBW
IN/OUT
IN/OUT
···
IN/OUT
CSW
dCBWDataTransferLength
Amount of data planned by the host
0 is returned because it is equal to the amount of data planned by the host
dCSWDataResidue
Amount of data planned by the device
dCBWDataTransferLength
Amount of data planned by the host
The amount of data insufficient for that planned by the host is returned
dCSWDataResidue
Amount of data planned by the device
Insufficient
dCBWDataTransferLength
Amount of data planned by the host
The amount of data exceeding that planned by the host is returned
dCSWDataResidue
Amount of data planned by the device
Exceeding

Figure 11 Each Stage in Bulk-Only Transport

6. Documents for Reference

- Software Manual
  (1) SH-2A, SH2A-FPU Software Manual (REJ09B0051)
  The most up-to-date version of this document is available on the Renesas Technology Website.
- Hardware Manual
  (2) SH7216 Group Hardware Manual (REJ09B0543)
  The most up-to-date version of this document is available on the Renesas Technology Website.
- USB standard-related
  (3) Universal Serial Bus Specification
  (4) Universal Serial Bus Mass Storage Class Specification Overview
  (5) Universal Serial Bus Mass Storage Class (Bulk-Only Transport)
  — Website for USB developers
  http://www.usb.org/developers

Website and Support

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

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

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	Jul.15.09	—	First edition issued

Notes regarding these materials