

Using the USB block as a Mass Storage Class device.....	2
1. Abstract.....	2
2. Introduction to USB	2
2.1 <i>Transfer types</i>	3
2.2 <i>Storage media and the FAT File System</i>	4
3. The Mass Storage Class	4
3.1 <i>Mass Storage Class Requirements:</i>	4
3.2 <i>The SCSI command Interface</i>	5
3.3 <i>The H8SX/1664 USB Peripheral</i>	5
4. Implementation.....	6
4.1 <i>State Transition Diagram</i>	6
4.2 <i>USB Communication State</i>	7
4.3 <i>File Structure</i>	8
4.4 <i>Purposes of Functions</i>	10
4.5 <i>RAM Disk</i>	13
5. Sample Program Operation.....	14
5.1 <i>Main Loop</i>	14
5.2 <i>Types of Interrupts</i>	15
5.3 <i>EPINFO</i>	16
5.4 <i>Bus Reset Interrupt (BRST)</i>	17
5.5 <i>Control Transfers</i>	17
5.6 <i>Bulk Transfers</i>	23
6. Using the program.....	30
7. Limitations	30
8. Data Sheet	30
9. References	30

Using the USB block as a Mass Storage Class device

1. Abstract

The following application note introduces the USB Mass Storage class and shows an example of how to configure the USB block on the H8SX/1664 and use the microcontroller as a Mass Storage class device. This document refers to the RSK H8SX/1664 kit and specifically to the included Mass Storage Class application example.

2. Introduction to USB

The USB (Universal Serial Bus) is an interface and a protocol that allows a single host computer to communicate with a variety of peripheral devices. The USB 2.0 spec defines this interface. Although it is dependant on the application most USB projects will require a host side interface app and the device firmware. Every USB communication is between a host and a device, where the host controls the bus and initiates communication all the time, except in case of devices with the remote-wakeup feature. In comparison with other interfaces, USB offers a host of advantages which include, automatic configuration (enumeration), minimum IRQ lines used, hot pluggable, low cost, low power consumption, speed and reliability. Depending on the application, the developers can chose one (or more) of four USB transfer types for the project; Control, Bulk, Interrupt and Isochronous. These classifications are based on frequency of transfer, amount of data to be transferred and the kind of data being transferred.

In USB terminology, individual devices are referred to as *functions*, which are linked in series through *hubs*. The hubs are special-purpose devices that are not considered functions. There always exists one hub known as the root hub, which is attached directly to the host controller.

Endpoints:

Functions and hubs have associated *pipes* (logical channels). Pipes are connections from the host controller to a logical entity on the device named an *endpoint*. The end point thus serves as a data buffer; typically it is a block of data memory or a register in the device each endpoint can transfer data in one direction only (except endpoint 0), either into or out of the device/function, so each pipe is unidirectional. Every device has endpoint zero configured for bidirectional control transfer. The number of available endpoints and supported transfer types vary with each device. The different kinds of endpoints are Bulk, Control, Interrupt and Isochronous. Since endpoints are unidirectional, they will be followed by an “in” or “out” specification (e.g. Bulk-In).

Device Information:

To identify itself as a USB device and to conform to the spec for a certain class, a device needs to have in its firmware certain elements of information that the host can access in order to successfully enumerate and then communicate with the device. These elements are broadly known as Descriptors and for the Mass Storage class they are further classified into:

- a) Device descriptor: Information such as the device class, the device sub-class, number of configurations, max packet size and other info about the device as a whole are present in this descriptor.
- b) Configuration Descriptor; Information about the number of interface supported and power consumption are provided in this descriptor; most devices usually support only a single configuration, but multiple configurations are allowed.

- c) Interface descriptor. (Specifying Mass Storage class): Each interface on the device has its own descriptor and subordinate descriptors (descriptors for endpoints used in the interface); the mass storage functionality is specified in this descriptor.
- d) Endpoint Descriptors (at least 2): Endpoint descriptors contain information about the endpoints to be used in that interface. This includes maximum packet size, polling rate, endpoint type (Interrupt, Bulk, Control or Isochronous) and endpoint direction (in or out).
- e) String Descriptor: Human readable information i.e. messages to be displayed on device enumeration etc are stored in this descriptor. It is optional.

Enumeration:

Before the host can begin using a USB device, it has to learn about the device capabilities, resources and other features in order to assign a device driver. The procedure by which a device identifies itself (including all resources and capabilities available) to the host is known as enumeration. When a function or hub is attached to the host controller through any hub on the bus (including the root hub), it is given a unique 7 bit address on the bus by the host controller. On any USB system all communication is initiated by the host. The host uses a specific set of requests to retrieve required information from the device. These requests can be classified as standard requests and class-specific requests. There are eleven standard requests in the USB.

The Get_Descriptor command is used to retrieve standard, class and vendor specific requests depending on the setting in the descriptor type field which is the high byte of the request. The Set_Descriptor request lets the host change descriptors in the device. The host controller then polls the bus for traffic, usually in a round-robin fashion, so no function can transfer any data on the bus without explicit request from the host controller.

USB establishes a 1 millisecond time base called a frame on a full-/low-speed bus. A frame can contain several transactions. Each transfer type defines what transactions are allowed within a frame for an endpoint. Isochronous and interrupt endpoints are given opportunities to access the bus every N frames. This information is set in the “*Interval*” or Polling Interval field in the endpoint descriptor. For Bulk endpoints, this field is not applicable

2.1 Transfer types:

Four transfer types are supported by the USB spec:

2.1.1 Control transfers: Control transfers are facilitated by the device control endpoint (endpoint zero). The host uses control transfers to configure the device, request device information and other settings. Control transfers are different from other transfers in that they have stages; typically three stages. The host sends a request in the Setup stage; the Data stage is used by the host/device to send data (not all requests have this stage) and the device reports the status information in the Status stage. Control transfers may also be used to send vendor specific requests.

2.1.2 Interrupt Transfers: Interrupt transfers are typically periodic communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data. These transfers require an Interrupt In endpoint on the device.

2.1.3 Bulk transfers: Bulk transfers can be used for large bursts of data. It is ideal in situations where the transfer rate is not critical. Data transfer using bulk transfers are very fast if the bus is idle; if the bus is busy, the transfers are delayed. This type of transfer is supported only by Full-Speed and High-Speed devices and

require a Bulk-In endpoint and a Bulk-Out endpoint for data to and from the PC respectively. This type of transfer is supported only by Full and high speed devices.

2.1.4 Isochronous transfers: Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. There is no retry or guarantee of delivery, although for the kind of application it is designed for, loss of a packet or frame does not cause critical issues with application performance e.g. audio or video glitches too small to be noticed by the user. The transfer mode is supported only by Full and High speed USB devices.

2.2 Storage media and the FAT File System:

There are many different kinds of storage media and they can be broadly grouped into hard drives and general flash memory from an embedded storage media point of view. While hard drives are preferred for larger capacities, flash memory is the media of choice in smaller capacity applications. While using flash memory, device firmware must implement “wear-leveling” in order to extend the life of the memory chip.

FAT file system:

The FAT (File Allocation Table) file system was created (and patented) by Microsoft for disk management purposes. The system essentially keeps a table of file names, location of contents, usable areas of memory etc. This file system is supported by virtually every OS, and hence is an ideal format in which to store data. The FAT file system is used extensively in embedded data storage systems that have to directly interface with an OS.

File system support is not a requirement for Mass Storage class devices but, dependant on the application, the developer may choose to implement this as well. If the firmware supports Logical Block Addressing, then any required data can be readily accessed; however, if there was need for this storage media to be useful in other environments outside that controlled by the device firmware (e.g. if the storage media were to be used in a Windows environment), then additional support/interface as required by that system will have to be provided. In most cases a file system provides the requisite interface.

In the provided application, a file diskimage.h includes the FAT file system; on startup, this image is copied into RAM, and thus the device enumerates as a preformatted FAT device.

For more information and articles on Embedded File Systems and Storage media refer to Jan Axelson’s USB Mass Storage page

3. The Mass Storage Class

USB devices are categorized in to various classes based on common behavior and protocols for devices that serve similar functions.

3.1 Mass Storage Class Requirements:

In addition to supporting standard USB requirements, a mass storage class device must conform to the following as well:

1. Interface Descriptor with a class code of 08h.
2. A mass storage interface with Bulk-in and Bulk-out end points and endpoint zero for control transfer.
3. Storage media (hard drive, flash-memory cards, CD/DVD, MMC cards etc).
4. Firmware support for logical block addressing (LBA) of the storage media.
5. Firmware support for Mass Storage class requests.

6. Support for one or more industry-standard command-block sets to exchange control, data and status information. (e.g. SCSI)

3.2 The SCSI command Interface

The SCSI (Small Computer Standard Interface) standard contains definitions of command sets of specific peripheral device types and using it to transfer data between computers and the specific peripheral device. However the presence of “unknown” as one of the device types, theoretically allows SCSI to be used to interface with practically any device.

Devices are classified based on the type of SCSI commands they support; the SCSI Block Commands (SBC) document specifies commands used by flash drives and other direct access block devices. In SCSI terminology, communication takes place between an initiator and a target. The initiator sends a command to the target which then responds. SCSI commands are sent in a Command Descriptor Block (CDB). The CDB consists of a one byte operation code followed by five or more bytes containing command-specific parameters. At the end of the command sequence the target returns a Status Code byte which is usually 00h for success, 02h for an error (called a Check Condition), or 08h for busy. When the target returns a Check Condition in response to a command, the initiator usually then issues a SCSI Request Sense command in order to obtain the status.

The total number of commands defined by the protocol are about 60; but how many are implemented is dependant on the application. At a minimum, dependant on the application, the following primary commands have to be implemented:

1. INQUIRY: This command is used by the host to request information about the device. The response from the device is in the form of a structure (at least 36 bytes in length) where information such as peripheral device type, vendor identification number, and other information about the devices capabilities is sent to the host. The response structure is sent in the data-transfer phase of the request.
2. READ CAPACITY: Tells the host the media sector information.
3. READ (10): Reads the specified sector volume data from a specified sector.
4. REQUEST SENSE: The host requests sense data via this command. In the event that the device experiences a problem, status information is filled into a structure; this data is called sense data.
5. TEST UNIT READY: This command is used to determine if the storage device is ready for use.
6. WRITE (10). Writes the specified sector volume data to a specified sector.

The following additional commands have been implemented by the sample program as well.

1. PREVENT/ALLOW MEDIUM REMOVAL: This command requests the device to prevent or allows removal of the storage media from the device. A 2 bit field is used to set/unset the option; support is optional for this command.
2. VERIFY: Verifies if the data in a medium can be accessed.
3. STOP/START UNIT: Controls installation and removal of media.
4. MODE SENSE (6): Tells the host the drive status.

3.3 The H8SX/1664 USB Peripheral

The H8SX CPU is a high-speed CPU with an internal 32-bit architecture that is upward compatible with the H8/300, H8/300H, and H8S CPUs.

The main features of the USB peripheral are:

- On-chip UDC (USB Device Controller) conforming to USB 2.0
- USB standard version 2.0 full-speed (12 Mbps) transfer supported
- Automatic processing of USB standard commands for endpoint 0. (Some commands need to be processed through the firmware)
- Four transfer modes supported (Control, Bulk, Interrupt and Isochronous)
- 16 interrupt signals
- On-chip bus transceiver
- Power mode: Self power mode or bus power mode can be selected by the power mode bit (PWMD) in the control register (CTLR).

Endpoint	Name	Transfer Type	Max Packet Size (bytes)	FIFO Capacity	Buffer	DMA Transfer
0	EP0s	Setup	8	8 bytes		
	EP0i	Control-in	8	8 bytes		
	EP0o	Control-out	8	8 bytes		
1	EP1	Bulk-Out	64	128 bytes		Available
2	EP2	Bulk-In	64	128 bytes		Available
3	EP3	Interrupt-in	8	8 bytes		

Table 3.1: Endpoint Configurations

Commands decoded by hardware	Commands not decoded by hardware
Clear Feature	Get descriptor
Get Configuration	Synch Frame
Get Interface	Set Descriptor
Get Status	Class/Vendor command
Set address	
Set Configuration	
Set Feature	
Set Interface	

Table 3.2: Standard USB command support

The applications included use commonly used USB functions and procedures. These functions and general code flow are described in the following sections.

4. Implementation

In this section, features of the sample program and its structure are explained. This sample program runs on the H8SX/1664, which works as a RAM disk, and initiates USB transfers by means of interrupts from the USB function module.

Features of this program are as follows.

- Control transfer can be performed.
- Bulk-out transfer can be used to receive data from the host controller.
- Bulk-in transfer can be used to send data to the host controller.
- It operates as a RAM disk that supports SCSI commands.

4.1 State Transition Diagram

Figure 4.1 shows a state transition diagram for this sample program. In this sample program, as shown in figure 4.1, there are transitions between four states.

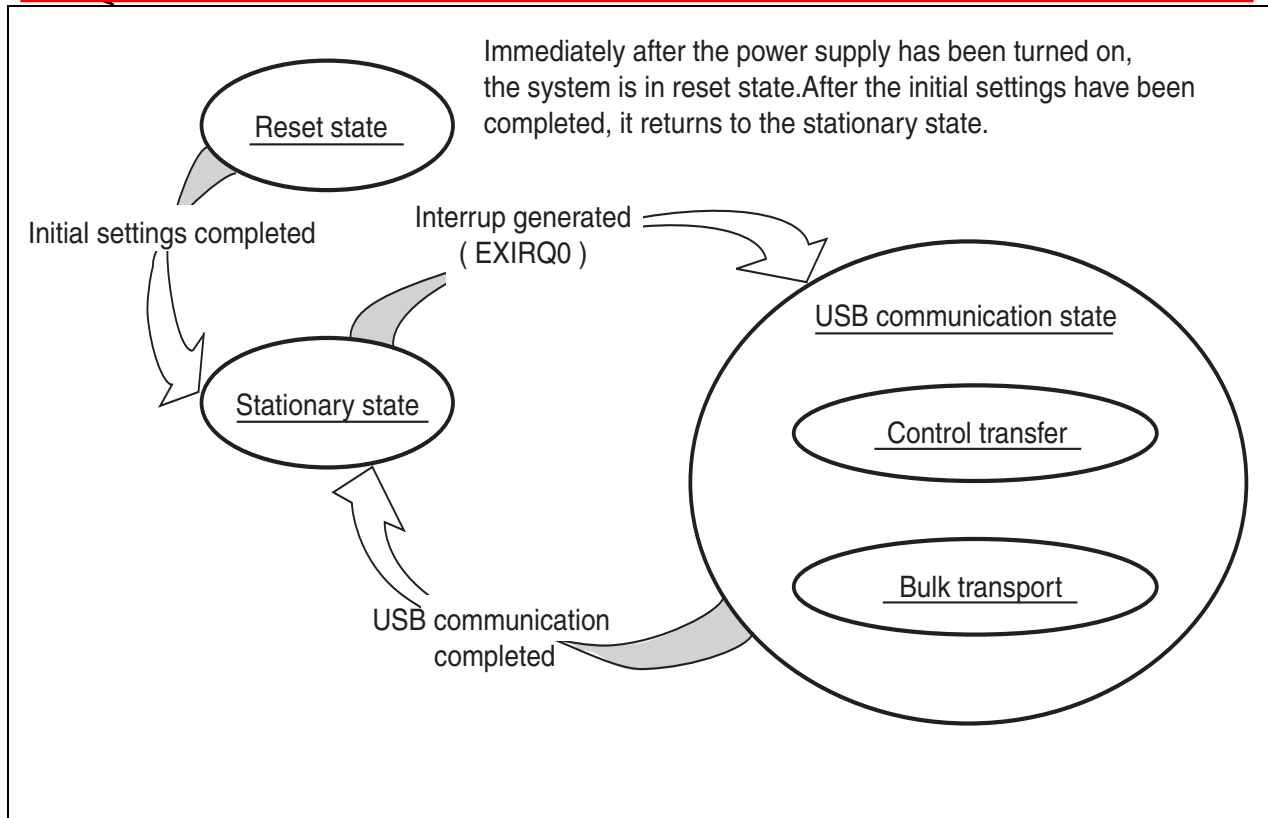


Figure 4.1: State Transition Diagram

- **Reset State**
Upon power-on reset and manual reset, this state is entered. In the reset state, the H8SX/1664 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 registers 0 (IFR0), and there are eight interrupt types in all. When an interrupt factor occurs, the corresponding bits in IFR0 are set to 1.

4.2 USB Communication State

The USB communication state can be further divided into two states according to the transfer type (see figure 4.2). 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. The branching method is explained in section 5, Sample Program Operation.

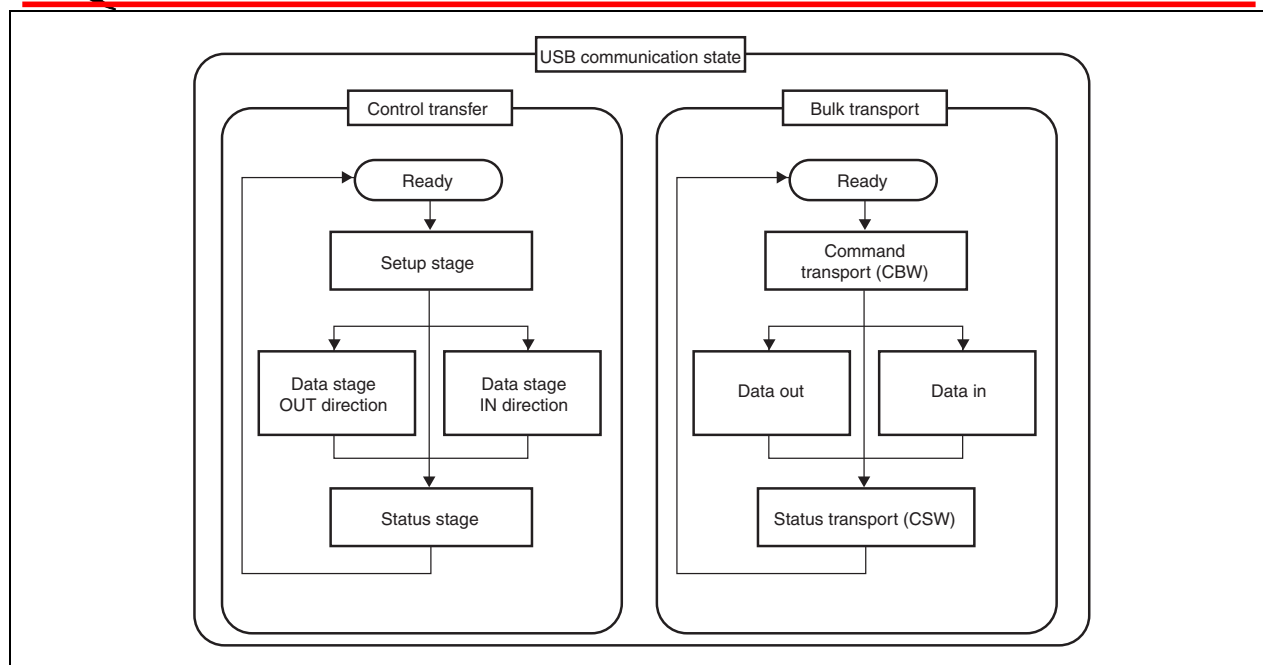


Figure 4.2: USB Communication State

4.2.1 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 transfer to be carried out.

Transfer 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.

4.2.2 Bulk Transfer

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

4.3 File Structure

This sample program consists of eight source files and eleven header files. The overall file structure is shown in table 4.1. Each function is arranged in one file by transfer method or function type. Figure 4.3 shows the layered configuration of these files.

File Name	Principle Role
StartUp.c	Microcontroller 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 Mass Storage Class (Bulk-Only Transport) class commands
DoControl.c	Executing control transfer
DoBulk.c	Executing bulk transfer
DoBOTMSClass.c	Executing Mass Storage Class (Bulk-Only Transport)
DoSCSICommand.c	Analyzing and processing SCSI commands
iodefine.h	Defining H8SX/1664 registers
SysMemMap.h	Defining H8SX/1664 memory map addresses
CatProType.h	Prototype declarations
CatTypeDef.h	Defining the basic structures used in USB firmware
CatBOTTypeDef.h	Defining structures used for Bulk-Only Transport
CatSCSITypedef.h	Defining structures used for SCSI and macros for preparing FAT information
SetUsbInfo.h	Default settings of variables needed to support USB
SetBOTInfo.h	Default settings of variables needed to support Bulk-Only Transport
SetSCSIInfo.h	Default settings of variables needed to support SCSI commands
SetSystemSwitch.h	System operation settings
SetMacro.h	Defining macros
sct.src	Specifying variables to be used to copy initial values from RAM

Table 4.1: File Structure

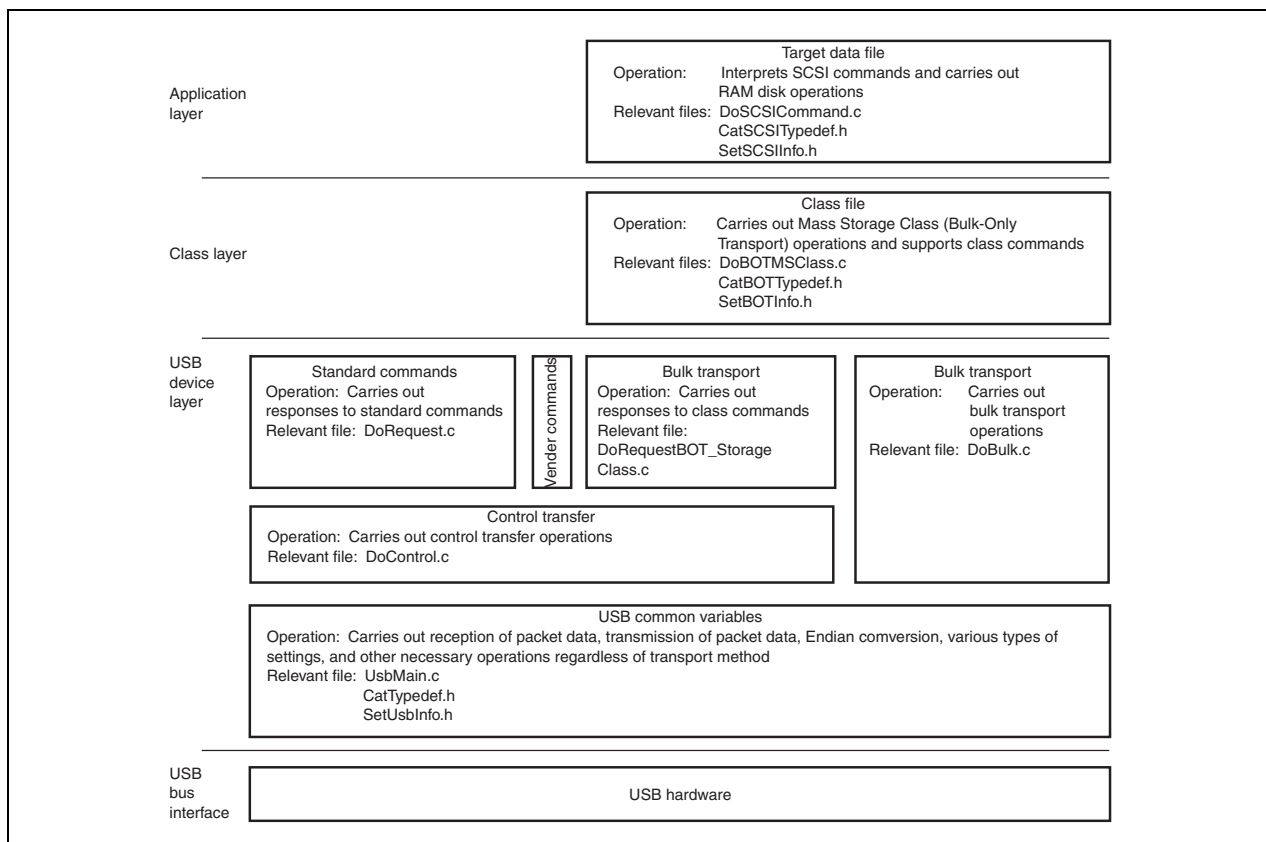


Figure 4.3: Layered Configuration of the Storage Class (BOT) firmware

4.4 Purposes of Functions

Table 4.2 to 4.9 shows functions contained in each file and their purposes.

File in Which Stored	Function Name	Purpose
StartUp.c	SetPowerOnSection	Sets BSC, terminals, and interrupt controller, calls initialization routines, and shifts to the main loop
	_INITSCCT	Copies variables that have default settings to the RAM work area
	InitMemory	Clears the RAM area used in bulk communication
	InitSystem	Specifies the USB clock, system interrupts, and masks
	SetEPInfpR	Write End Point information

Table 4.2: StartUp.c

When a power-on reset or manual reset is carried out, the SetPowerOnSection of the StartUp.c file is called and the H8SX/1664 default settings are entered.

File in Which Stored	Function Name	Purpose
UsbMain.c	BranchOfInt	Discriminates interrupt factors, and calls function according to interrupt
	GetPacket	Writes data transferred from the host controller to RAM
	GetPacket4	Writes data transferred from the host controller to RAM in longwords (This function will not use in this CPU)
	GetPacket4S	Writes data transferred from the host controller to RAM in longwords (This function will not use in this CPU)
	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(This function will not use in this CPU)
	PutPacket4S	Writes data for transfer to the host controller to the USB module in longwords(This function will not use in this CPU)
	SetControlOutContents	Overwrites data with that sent from the host
	SetUsbModule	Sets USB module initial settings
	ActBusReset	Clears FIFO on receiving bus reset
	ActBusVcc	Pulls up D+ and controls USB module when the USB cable is connected or disconnected (not used by this sample application)
	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

Table 4.3: 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.

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

Table 4.4: DoRequest.c

During control transfer, commands sent from the host controller are decoded and processed. In this sample program, a vendor ID of 045B (vendor: Renesas Technology Corp.) 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, DecVendorCommands does not perform any action. In order to use a vendor command, the customer should develop a program and handle it in the DecVendorCommands function.

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

Table 4.5: DoRequestBOT_StorageClass.c

This function carries out processing according to the 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.

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 transfer (transfer in which the data stage is in the IN direction)
	ActControlOut	Controls the data stage and status stage of control OUT transfer (transfer 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.

Table 4.6: 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.

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

Table 4.7: DoBulk.c

These functions carry out processing involving bulk transfer.

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)

Table 4.8: DoBOTMSClass.c

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

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

Table 4.9: 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.

Figure 4.4 shows the interrelationship between the functions explained in table 4.2 to 4.9. The upper-side functions can call the lower-side functions. Also, multiple functions can call the same function. In the stationary state, SetPowerOnSection calls other functions, and in the case of a transition to the USB communication state which occurs on an interrupt, BranchOfInt calls other functions. Figure 4.4 shows the hierarchical relation of functions; there is no order for function calling. For information on the order in which functions are called, please refer to the flow charts of section 5, Sample Program Operation.

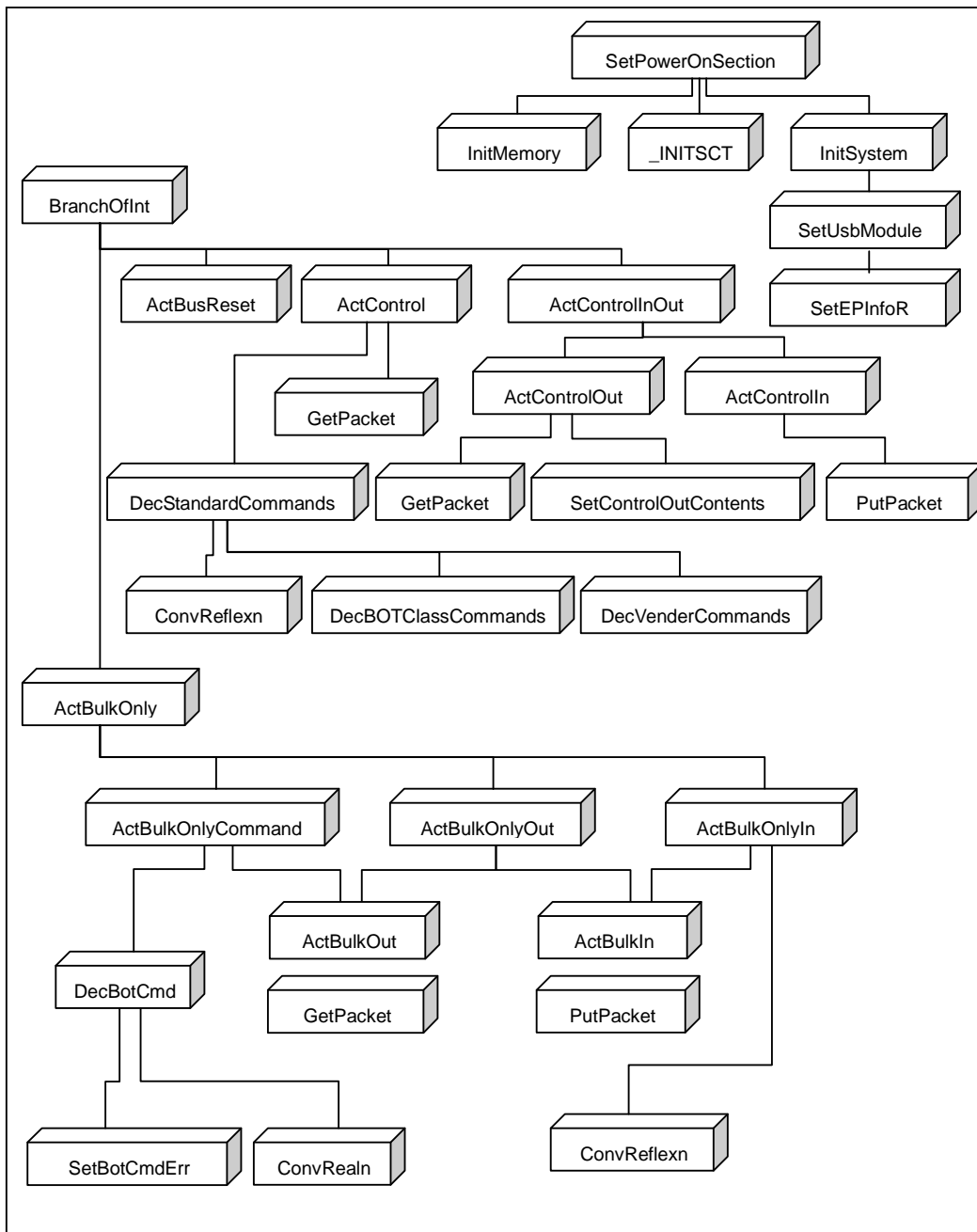


Figure 4.4: Interrelationships between Functions

4.5 RAM Disk

In the sample program provided here, the Internal RAM in the H8SX/1664 is selected as the disk device, and the host PC is notified that the H8SX/1664 (function) is a disk.

As shown in figure 4.5, 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 Internal RAM.

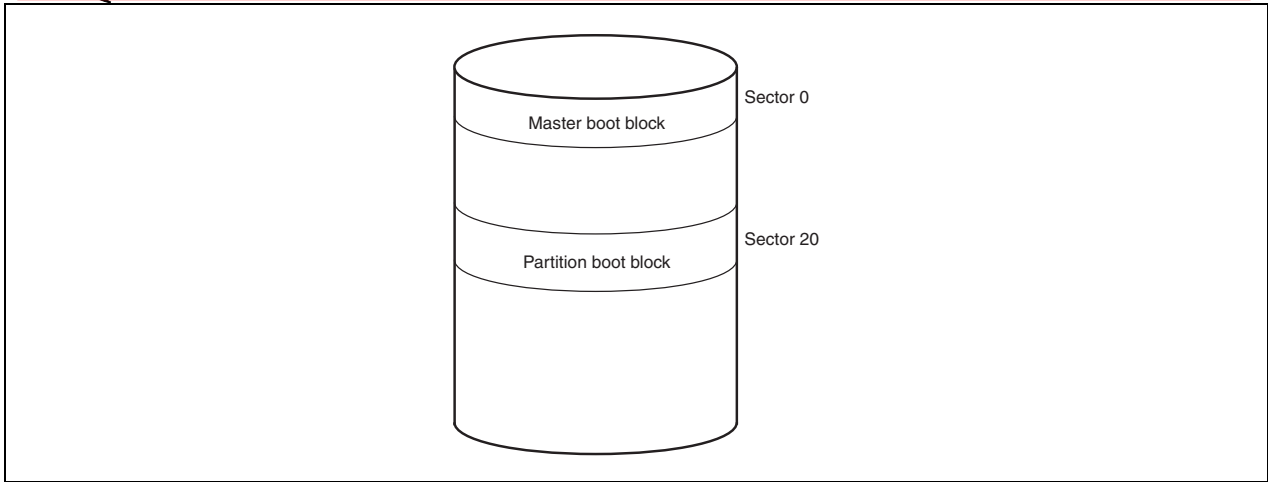


Figure 4.5: Disk Construction

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 4.5 and then write the operation.

5. Sample Program Operation

In this chapter, the operation of the sample program is explained, relating it to the operation of the USB function module.

5.1 Main Loop

When the microcomputer is in the reset state, the internal state of the CPU and the registers of internal peripheral modules are initialized. Next, the function SetPowerOnSection in StartUp.c is called, and the CPU is initialized. Figure 5.1 is a flow chart for the SetPowerOnSection function operation.

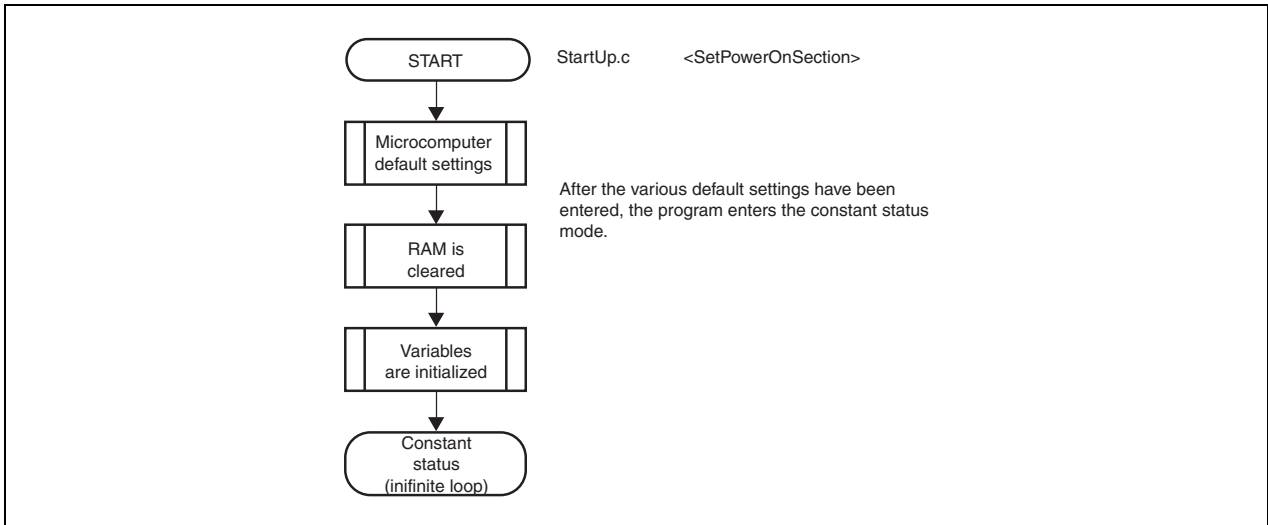
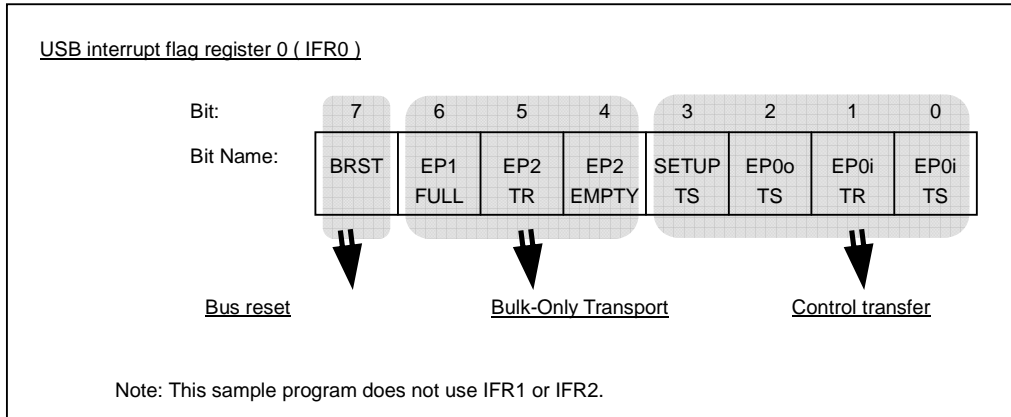


Figure 5.1 Main Loop

5.2 Types of Interrupts

As explained in section 4, the interrupts used in this sample program are indicated by the interrupt flag registers 0 (IFR0); there are a total of eight types of interrupts. When an interrupt factor occurs, the corresponding bits in the interrupt flag registers are set to 1, and an USBINTN2 interrupt request is sent to the CPU. In the sample program, the interrupt flag registers are read as a result of this interrupt request, and the corresponding USB communication is performed. Figure 5.2 shows the interrupt flag registers and their relation to USB communication.

Figure 5.2 Types of Interrupt Flags



5.2.1 Method of Branching to Different Transfer Processes

In this sample program the transfer method is determined by the type of interrupt from the USB module. Branching to the different transfer methods is executed by BranchOfInt in UsbMain.c. Table 5.1 shows the relations between the types of interrupts and the functions called by BranchOfInt.

Register Name	Bit	Bit Name	Name of Function Called
IFR0	7	BRST	ActBusReset
	6	EP1 FULL	ActBulkOnly
	5	EP2 TR	ActBulkOnly
	4	EP2 EMPTY	ActBulkInReady
	3	SETUP TS	ActControl
	2	EP0o TS	ActControlInOut
	1	EP0i TR	ActControlInOut
	0	EP0i TS	ActControlInOut

Table 5.1 Interrupt Types and Functions Called on Branching

The EP0iTS and EP0oTS interrupts are used both for control-in and control-out transfer. Hence in order to manage the direction and stage of control transfer, the sample program has three states: TRANS_IN, TRANS_OUT, and WAIT. For details, refer to section 5.6, Control Transfers.

In the H8SX/1664 hardware manual, operation of the USB function module when an interrupt occurs and a summary of operation on the application side are described. From the next section, details of application-side firmware are explained for each USB transfer method.

5.3 EPINFO

When initializing it, the USB function module equipped with H8SX1653 should set the End Point configuration with software. The forwarding type that can be set is shown as follows.

- Control transfer :1 pipe
- BulkIn transfer :1 pipe
- BulkOut transfer :1 pipe
- InterruptIn transfer :1 pipe

Each End Point is possible to set Interface Number, Alternate Number and MaxPacketSize by using End Point Information Resister (henceforth EPIR).

This sample program uses following End Point Configuration (Figure 5.3).

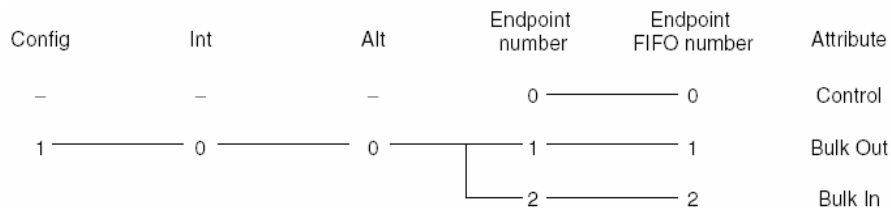


Figure 5.3 End Point Configuration

A setting value of EPIR that achieves the EndPoint composition shown in Figure 5.3 is indicated in Table 5.2. Please see the H8SX1653 hardware manual for details.

EPIR	Setting value (Hexadecimal)	Transfer type	EP Number	Conf.	Int.	Alt.	MaxPacket Size(Byte)	EPPIFO Number
0	00_00_10_00_00	Control	0	-	-	-	8 Byte	0
1	14_20_80_00_01	Bulk OUT	1	1	0	0	64 Byte	1
2	24_28_80_00_02	Bulk IN	2	1	0	0	64 Byte	2
3	00_00_00_00_00	-	-	-	-	-	-	-
4	00_00_00_00_00	-	-	-	-	-	-	-
5	00_00_00_00_00	-	-	-	-	-	-	-
6	00_00_00_00_00	-	-	-	-	-	-	-
7	00_00_00_00_00	-	-	-	-	-	-	-
8	00_00_00_00_00	-	-	-	-	-	-	-
9	00_00_00_00_00	-	-	-	-	-	-	-
10	00_00_00_00_00	-	-	-	-	-	-	-

Table 5.2: Set value of EPIR

5.4 Bus Reset Interrupt (BRST)

When the host controller detects that a device has been connected to the USB data bus, it outputs a bus reset signal. When receiving this bus reset signal, the USB function module generates a bus reset.

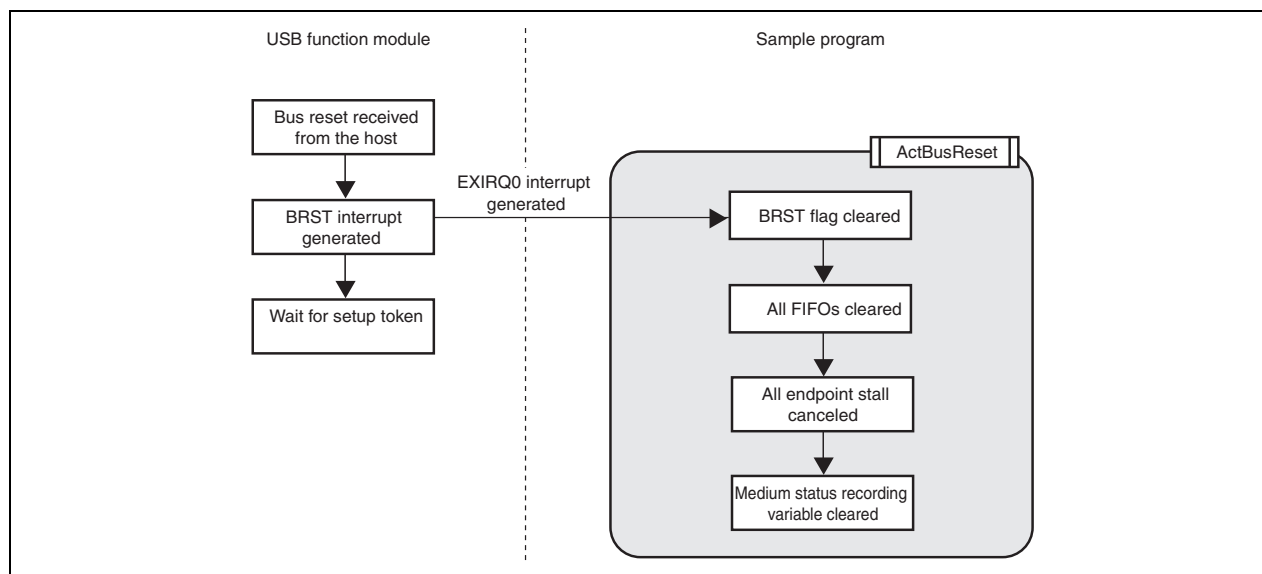


Figure 5.4 Bus Reset Interrupt

5.5 Control Transfers

In control transfers, bits 0 to 3 of the interrupt flag registers are used. Control transfers can be divided into two types according to the direction of data in the data stage. (Figure 5.5) In the data stage, data transfers from the host controller to the USB function module are control-out transfers, and transfers in the opposite direction are control-in transfers.

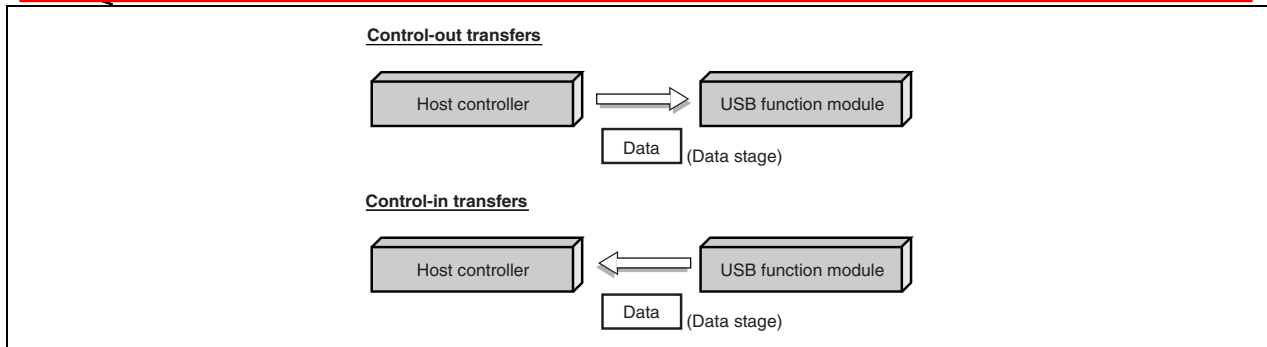


Figure 5.5 Control Transfers

Control transfers consist of three stages: setup, data (no data is possible), and status (figure 5.6). Further, the data stage consists of multiple bus transactions.

In control transfers, stage changes are recognized through the reversal of the data direction. Hence the same interrupt flag is used to call a function to perform control-in or control-out transfers (table 5.1). For this reason, the firmware must use states to manage the type of control transfer currently being performed, whether control-in or control-out, (figure 5.6) and must call the appropriate function. States in the data stage (TRANS_IN and TRANS_OUT) are determined by commands received in the setup stage.

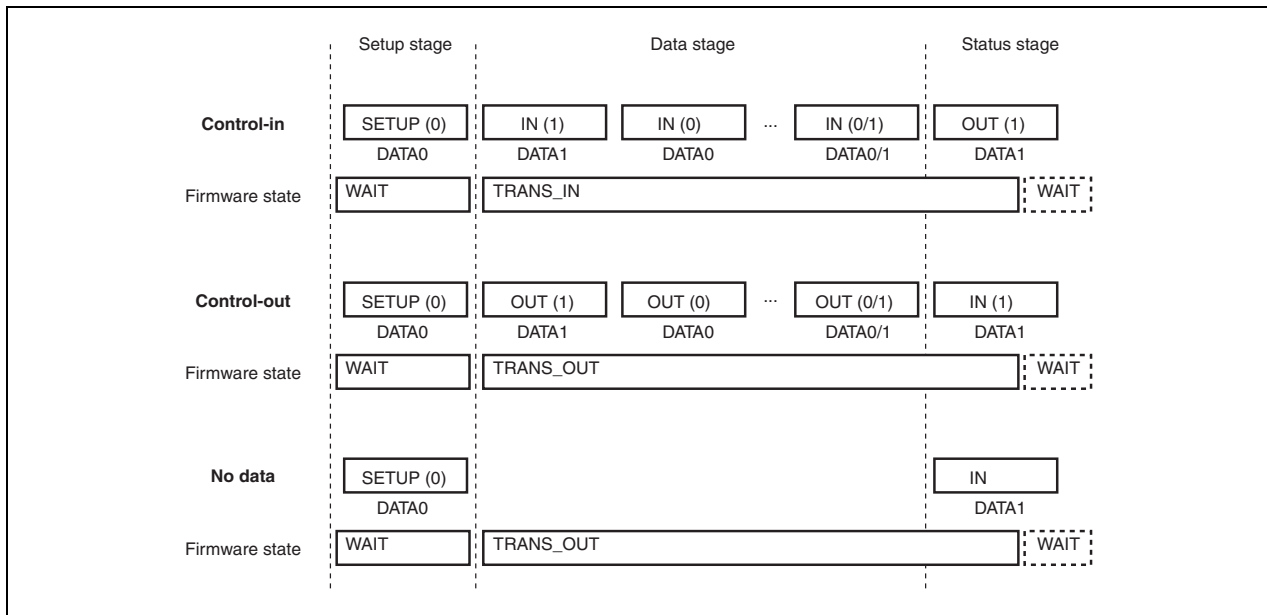


Figure 5.6: Status in Control Transfers

5.5.1 Setup Stage

In the setup stage, the host and function modules exchange commands. For both control-in and control-out transfer, the firmware goes into the WAIT state. Depending on the type of command issued, discrimination between control-in transfer and control-out transfer is performed, and the state of the firmware in the data stage (TRANS_IN or TRANS_OUT) is determined.

- Commands for control-in transfers: GetDescriptor (Standard command), Get Max LUN (Class command)
- Commands for control-out transfers: Bulk-Only Mass Storage Reset (Class command)

Figure 5.7 shows operation of the sample program in the setup stage. The figure on the left shows operation of the USB function module.

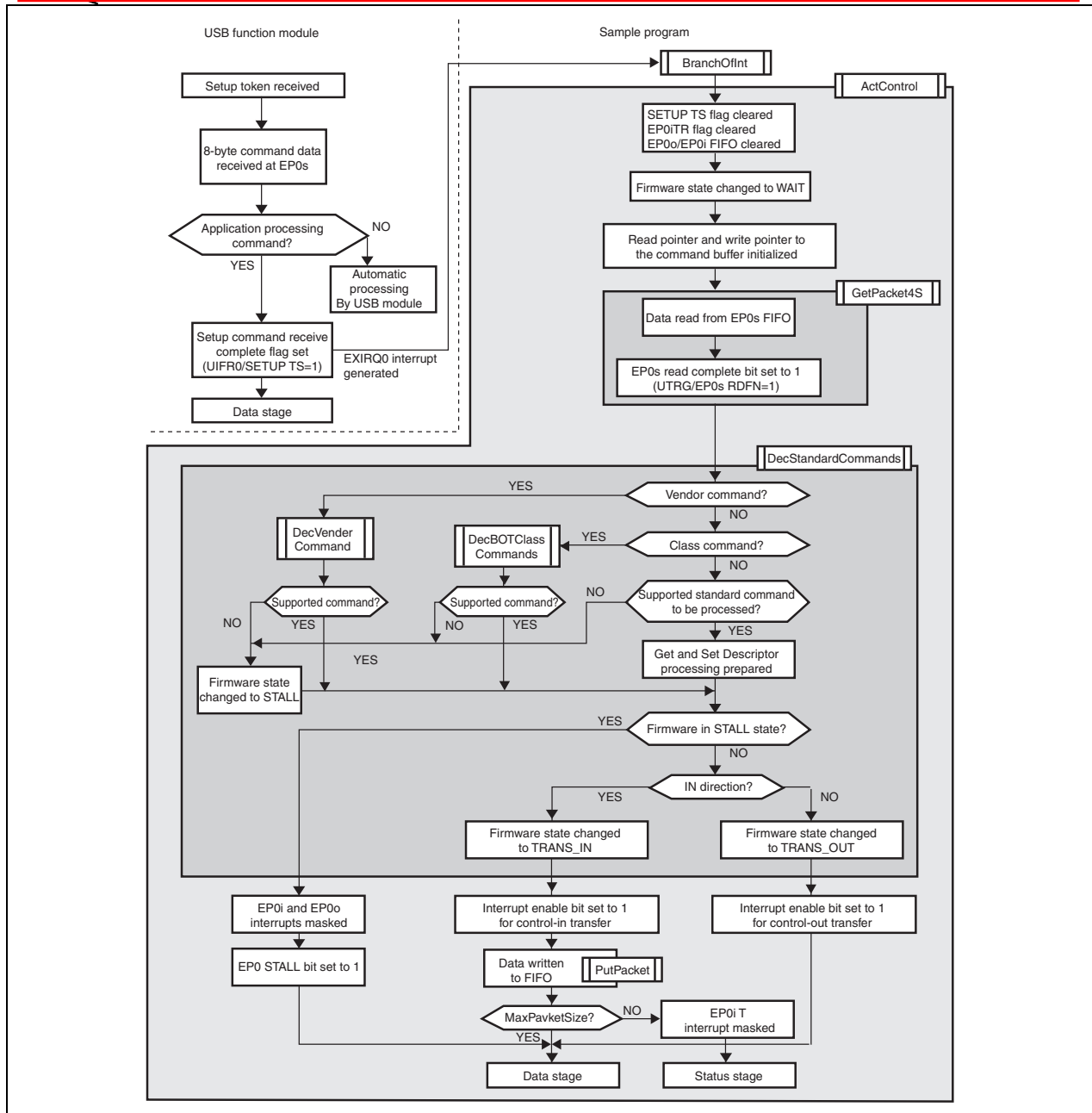


Figure 5.7 Setup Stage

5.5.2 Data Stage

In the data stage, the host and function module exchange data. The firmware state becomes TRANS_IN for control-in transfers and TRANS_OUT for control-out transfers, according to the result of decoding of the command in the setup stage. Figures 5.8 and 5.9 show the operation of the sample program in the data stage of control transfer.

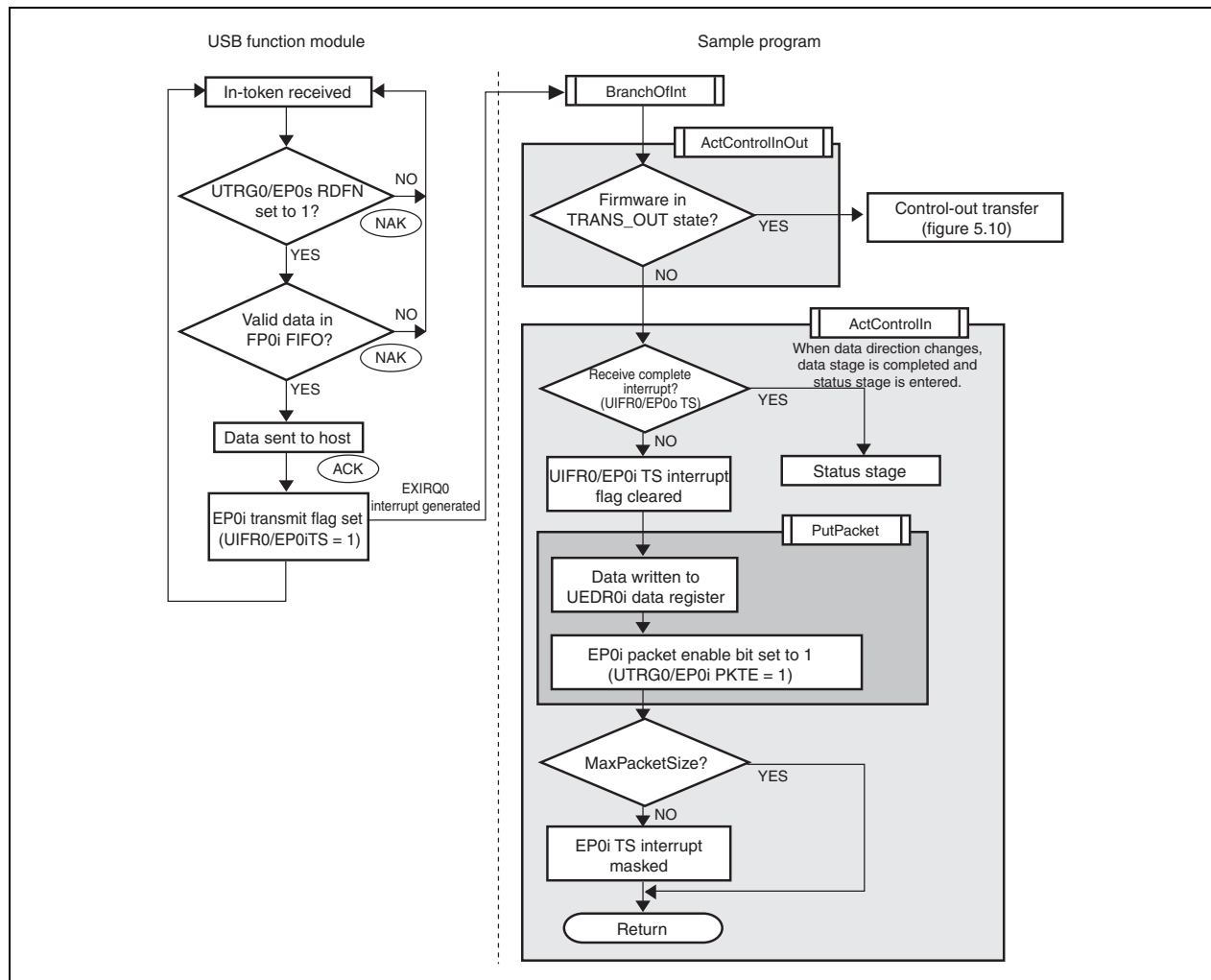


Figure 5.8 Data Stage (Control-In Transfer)

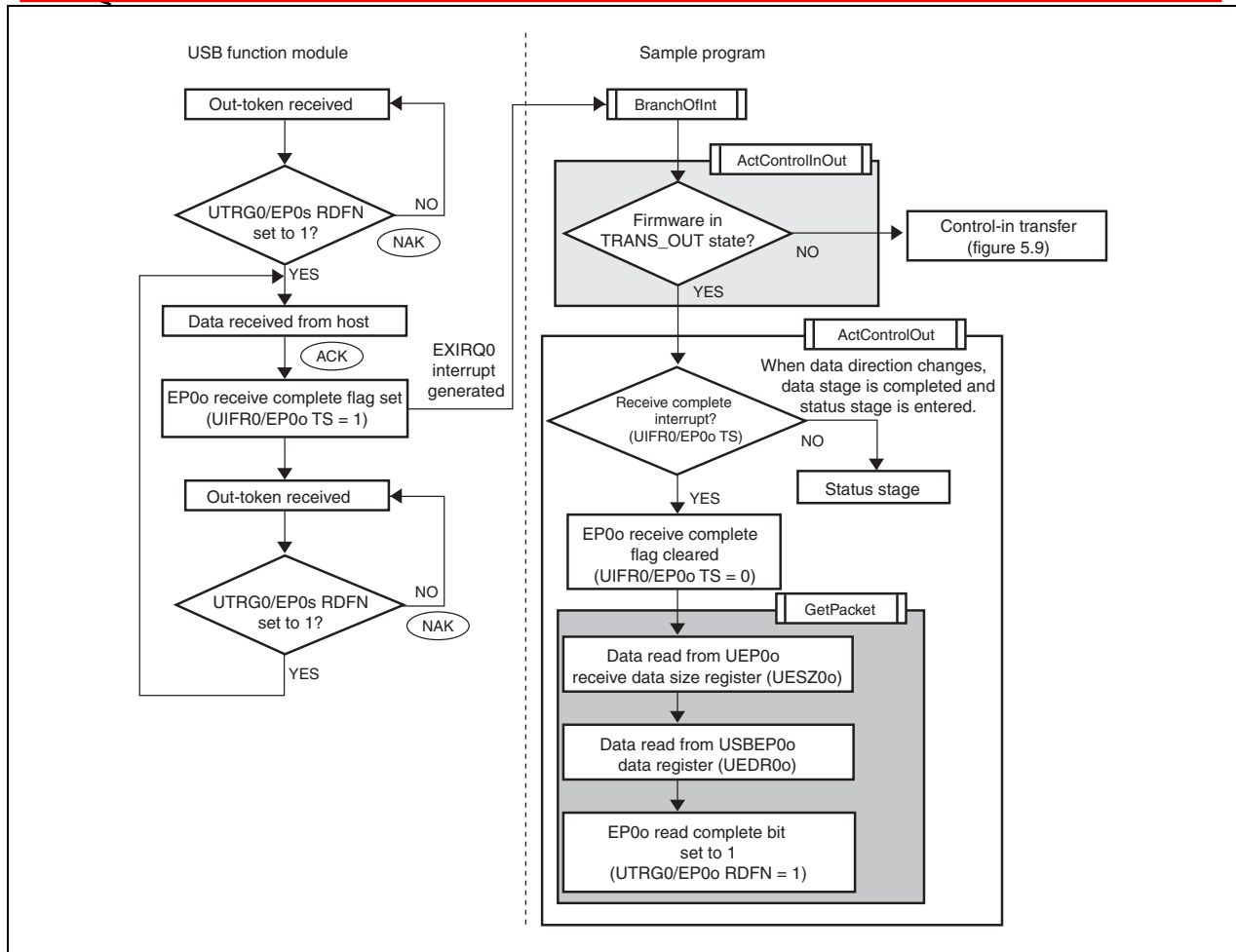


Figure 5.9 Data Stage (Control-Out Transfer)

5.5.3 Status Stage

The status stage begins with a token for the opposite direction from the data stage. That is, in control-in transfer, the status stage begins with an out-token from the host controller; in control-out transfer, it begins with an in-token from the host controller.

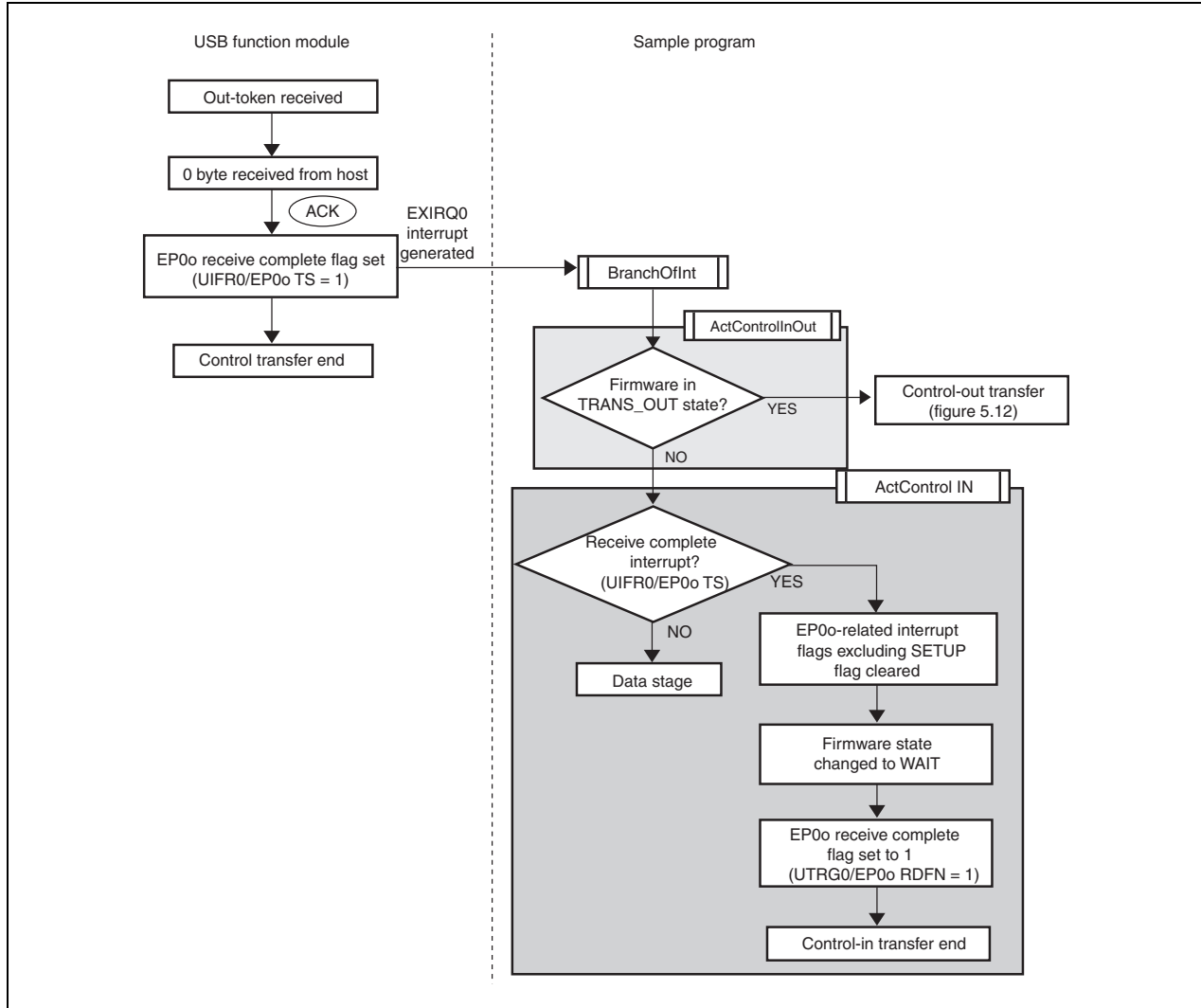


Figure 5.10 Status Stage (Control-In Transfer)

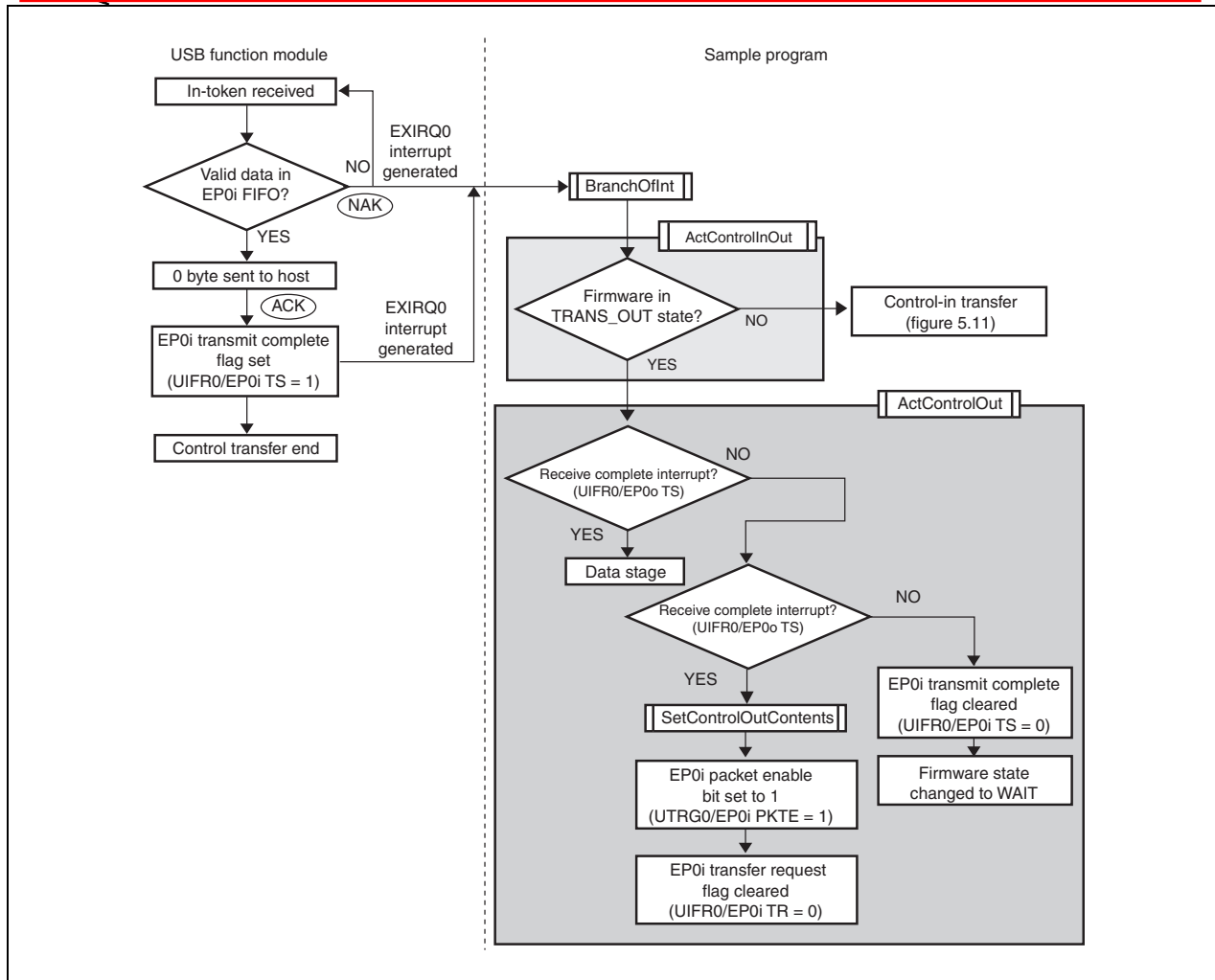


Figure 5.11 Status Stage (Control-Out Transfer)

5.6 Bulk Transfers

In bulk transfers, bits 0 to 2 of interrupt flag register 1 are used. Bulk transfers can also be divided into two types according to the direction of data transmission. (Figure 5.12)

When data is transferred from the host controller to the USB function module, the transfer is called a bulk-out transfer; when data is transferred in the opposite direction, it is a bulk-in transfer.

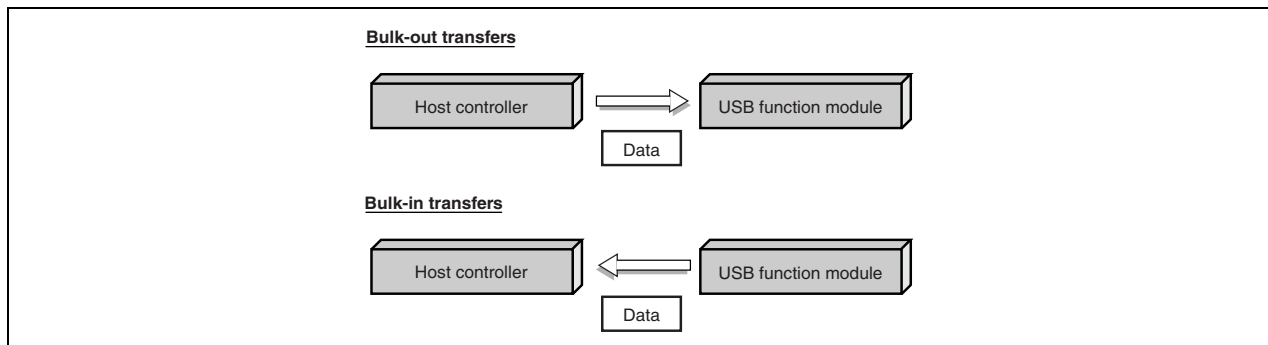


Figure 5.12 Bulk Transfers

The Bulk-Only Transport used in the USB Mass Storage Class consists of bulk-in and bulk-out transfers. Bulk-Only Transfer comprises two or three stages (figure 5.13): command transport (CBW), data transport (this is sometimes not included), and status transport (CSW). In addition, data transfer is made up of multiple bus

transactions.

With Bulk-Only transport, the command transport (CBW) is done using bulk-out transfer, while the status transport (CSW) is done using bulk-in transfer. Either bulk-in transfer or bulk-out transfer may be used for data transport, depending on the direction in which the data is being sent.

Whether bulk-in or bulk-out transfer is used for data transport is determined by the CBW data received using command transport. In the firmware, whether bulk-in or bulk-out is used for data transport is controlled by states (TRANS_IN and TRANS_OUT) (figure 5.13). The appropriate functions must be called by the firmware.

Additionally, the transition in stages from data transport to status transport is handled by data of a planned length being sent or received using data transport requested by the host PC. Consequently, the firmware manages the data length sent or received using data transport, and after the transition between stages, status transport must be used to send the data to the host PC.

If the CBW data received using command transport cannot be acknowledged as valid, the endpoint is stalled, and no bulk transfer is carried out.

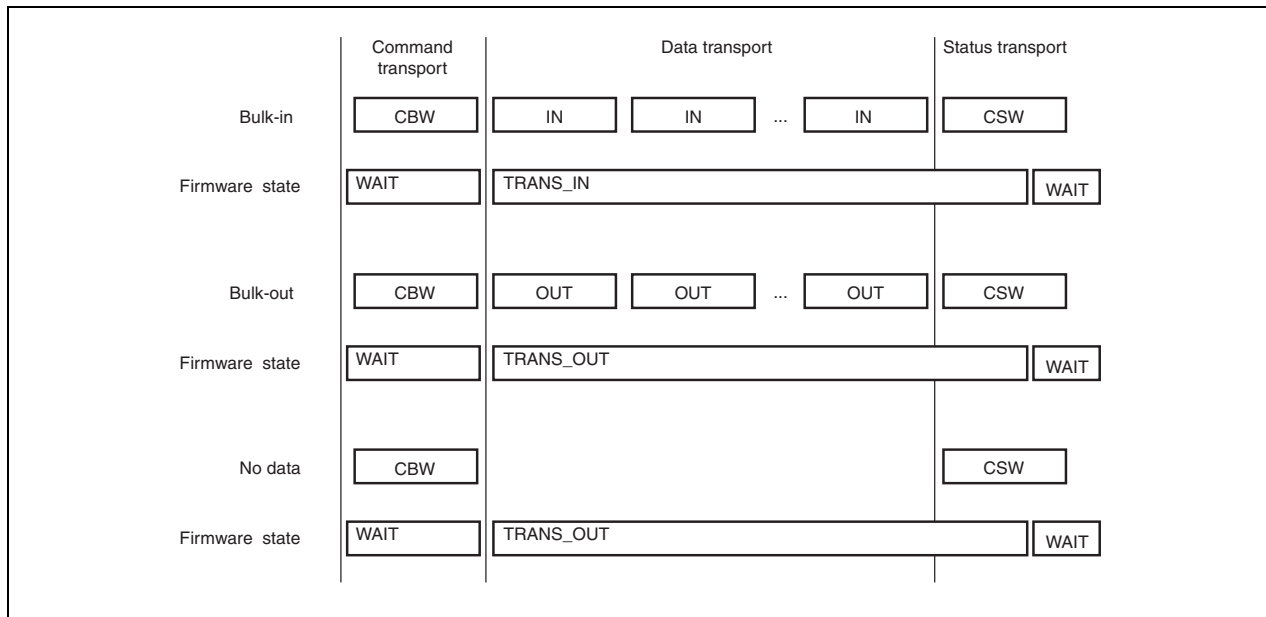


Figure 5.13 Various Stages in Bulk-Only Transport

5.6.1 Command Transport

With command transport, the CBW data is transferred from the host to the function.

At this point, the firmware is in the WAIT state. At the stage following reception of the CBW data, the five types of processing listed below are carried out.

1. The CBW data is stored from the EP1 data register to the work area.
2. A judgment is made as to whether the CBW data is valid.
3. The CSW data is prepared.
4. The contents of the CBW data are decoded, and if there is any data to be sent using data transport, the data is prepared. (Processing is carried out in the DecBotCmd function.)
5. A distinction is made as to whether the data transport is bulk-in or bulk-out, and the firmware state (TRANS_IN or TRANS_OUT) is determined.

Figure 5.14 shows the operation carried out by the sample program when command transport is used. The operation of the USB function module is shown at the left of the illustration.

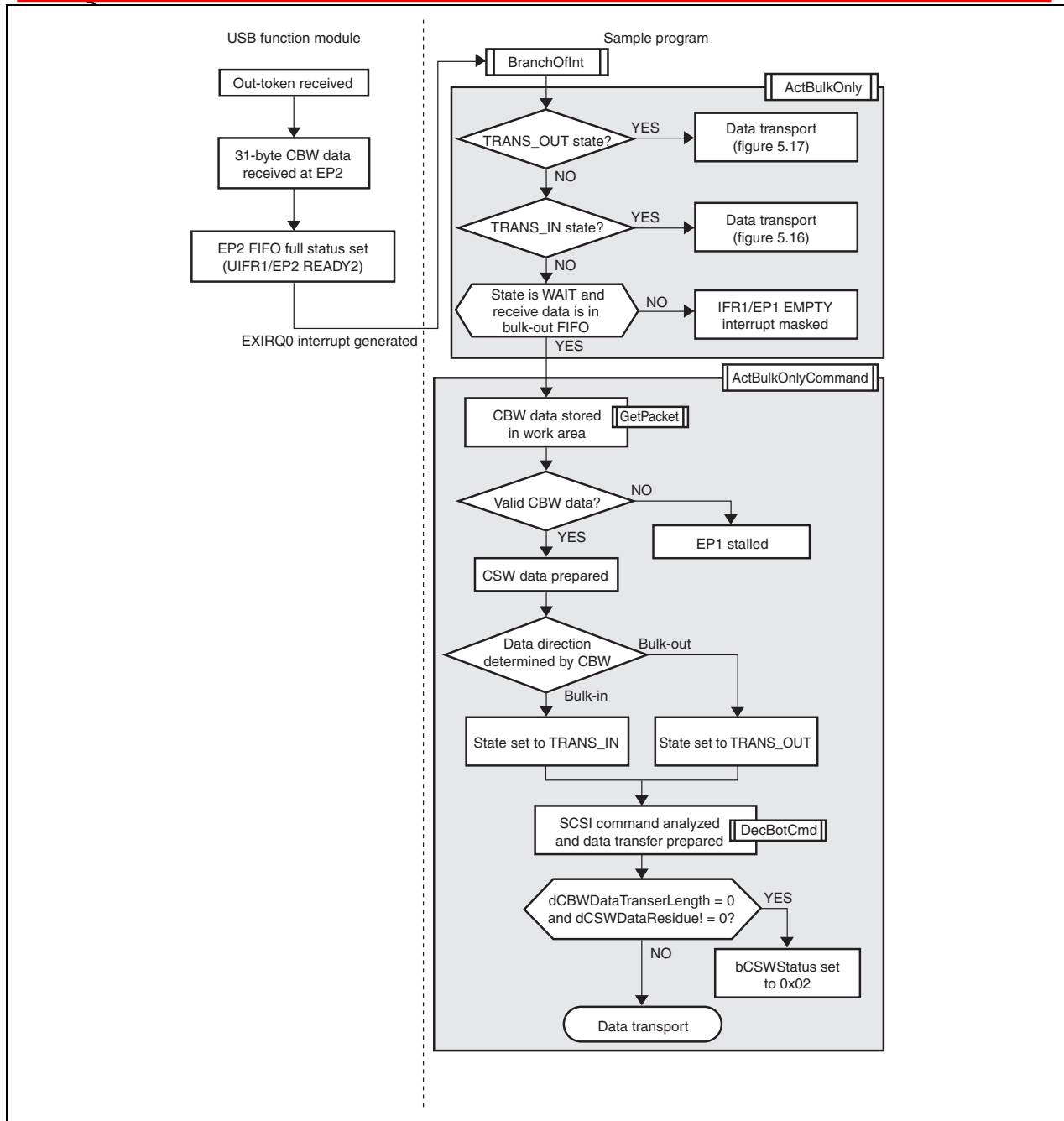


Figure 5.14 Command Transport

5.6.2 Data Transport

With data transport, data is sent and received between the host and the function.

At this point, the firmware is in either the TRANS_IN or TRANS_OUT state.

If the firmware state is TRANS_IN (bulk-in transfer), the following three types of processing are carried out.

1. Data is sent from the function to the host.
2. If the length of the data sent by the function is shorter than the length planned by the host, 0 is added.
3. The information to be sent by the CSW is created.

Figure 5.15 shows the operations that take place when data transport (bulk-in transfer) is carried out in the sample program. The operation of the USB function module is shown at the left side of the illustration.

In this sample software, if the length of the data sent by the function is shorter than the length of the data

requested by the host, 0 is added after the data sent by the function, as noted in the Bulk-Only Transport of the USB Mass Storage Class, and after data of the length requested by the host has been sent, the number of 0 bytes added is reported, using status transport.

In order to carry out this operation, the following is used as global variables: the `dCBWDataTransferLength` of the CBW data, the `dCSWDataResidue` of the CSW data, and the `bCSWStatus` of the CSW data.

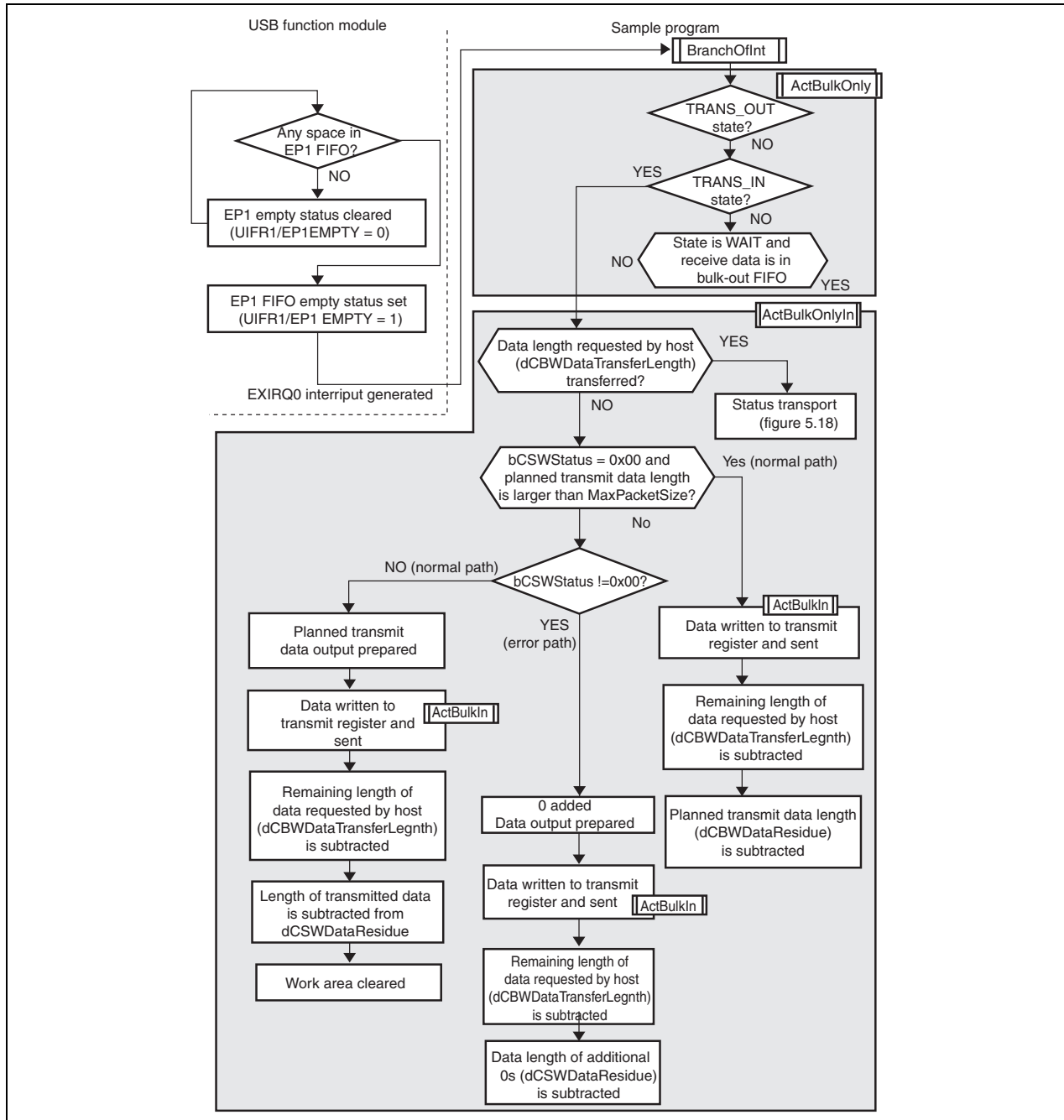


Figure 5.15 Data Transport (Bulk-In Transfer)

Figure 5.16 shows the operations that take place when data transport (bulk-out transfer) is carried out. If the firmware state is `TRANS_OUT` (bulk-out transfer), the following three types of processing are carried out.

1. Data from the host is received in the function.
2. Data length is calculated.
3. The information to be sent by the CSW is created.

In this sample software, if the length of the data received by the function is shorter than the length of the data

that the host planned to send, the missing length of data received by the function using data transport is reported using status transport. In order to carry out this operation, the following is used as global variables: the `dCBWDataTransferLength` of the CBW data and the `dCSWDataResidue` of the CSW data.

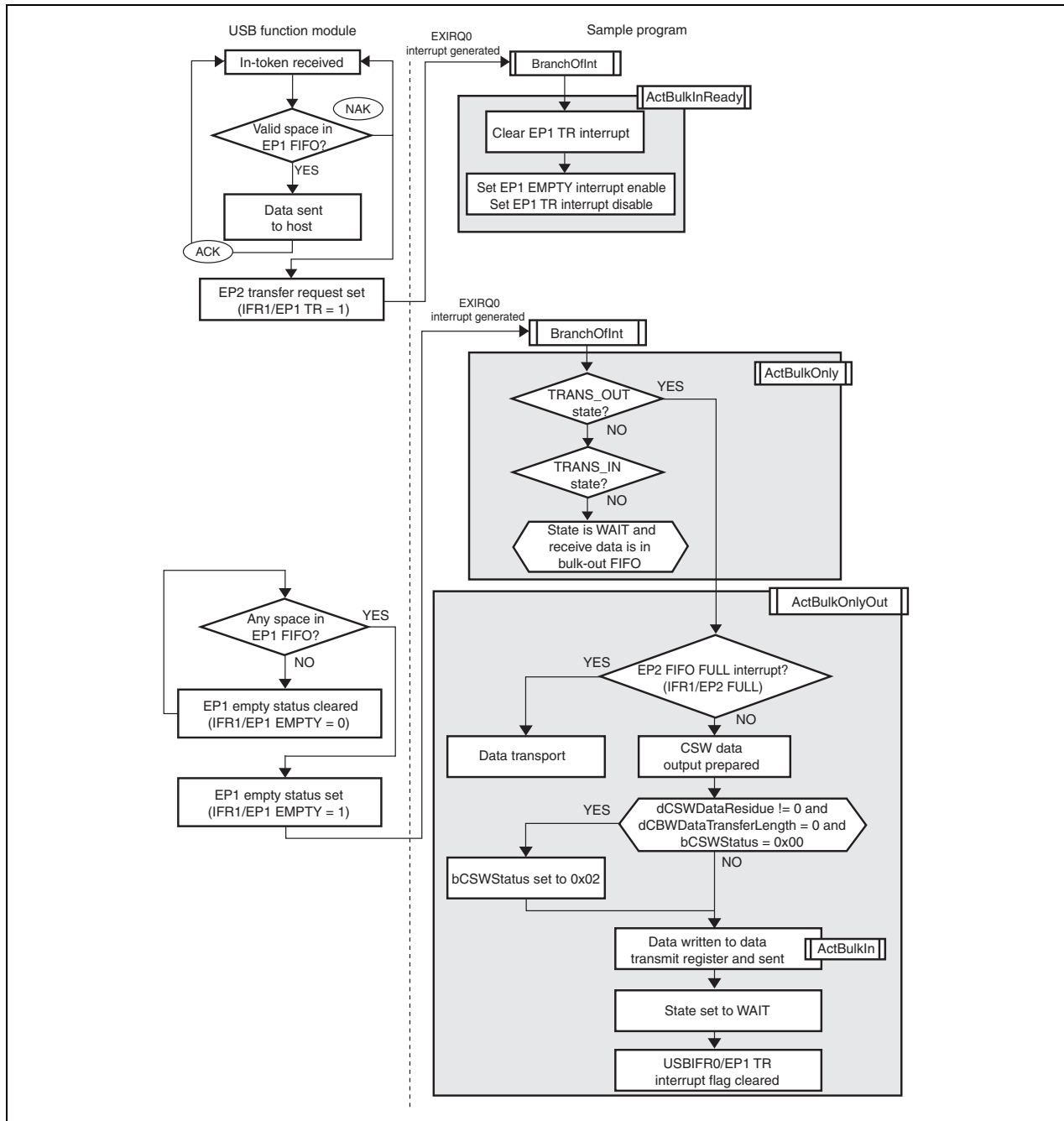


Figure 5.16 Data Transport (Bulk-Out Transfer)

5.6.3 Status Transport

With status transport, data is sent from the function to the host.

At this point, the firmware is in either the `TRANS_IN` or `TRANS_OUT` state.

If the firmware state is `TRANS_IN` (bulk-in transfer), the following four types of processing are carried out.

1. EP1 empty status interrupts are inhibited.
2. The system prepares to send the CSW data.
3. The CSW data is issued.

4. The firmware state is set to WAIT.

Figure 5.17 shows the operations that take place when status transport (data transport bulk-in transfer) is carried out in the sample program. The operation of the USB function module is shown at the left side of the illustration.

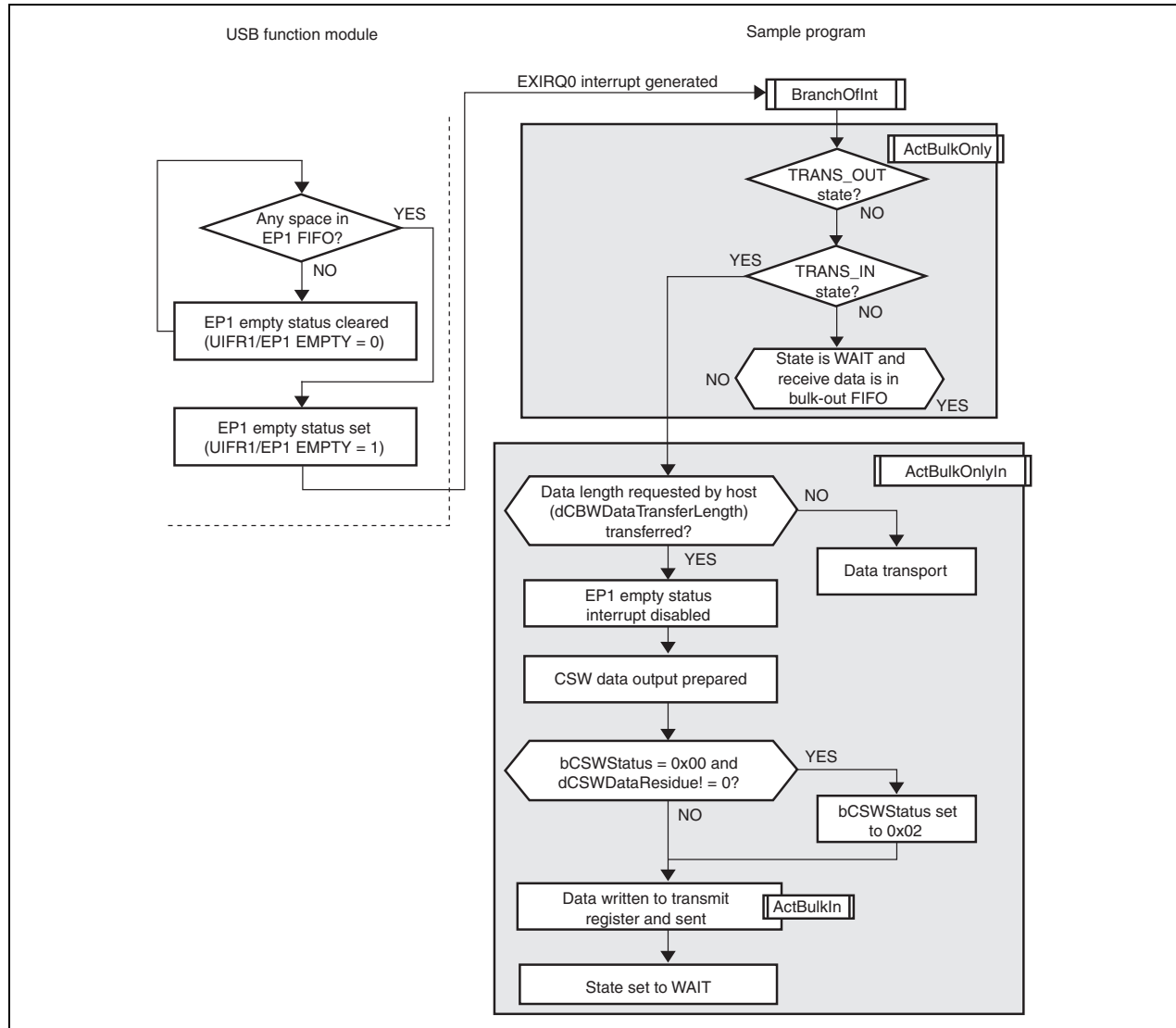


Figure 5.17 Status Transport (Data Transport Bulk-In Transfer)

Figure 5.18 shows the operations that take place when status transport (data transport bulk-out transfer) is carried out in the sample program. The operation of the USB function module is shown at the left side of the illustration.

If the firmware state is TRANS_OUT (bulk-out transfer), the following four types of processing are carried out.

1. Preparation is made to send the CSW data.
2. The data is checked to see if any data is missing from the reception.
3. The CSW data is issued.
4. The firmware state is set to WAIT.

In this sample software, if the length of the data received by the function is shorter than the length of the data that the host planned to send, the missing length of data received by the function using data transport is reported using status transport, as noted in the Bulk-Only Transport of the USB Mass Storage Class. In order to do this, a check is made to see if there is any data missing that should have been received by the function, and if there is, the value of the bCSWStatus of the CSW data is set to 0x02 (phase error).

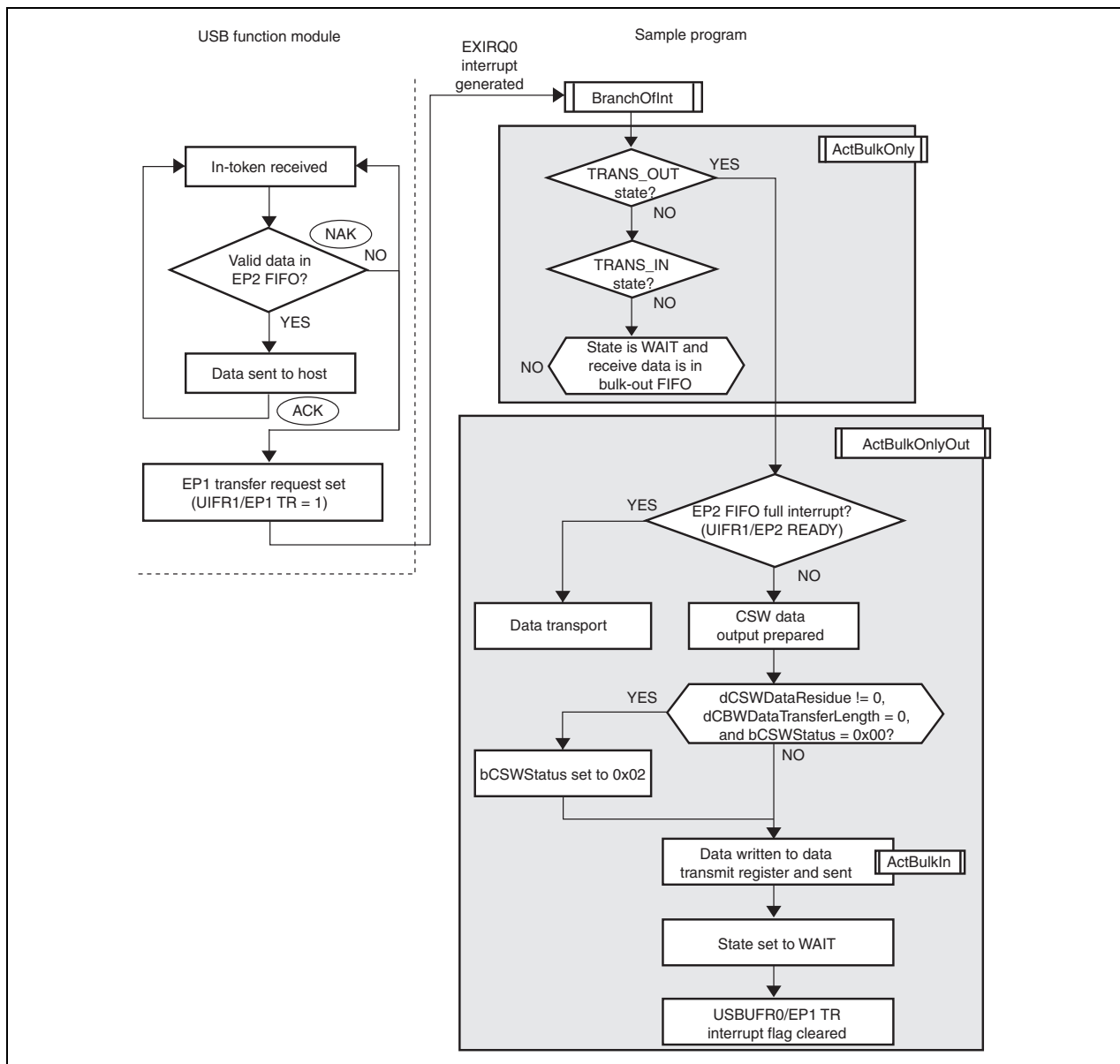


Figure 5.18 Status Transport (Data Transport Bulk-Out Transfer)

6. Using the program

The sample program has been configured to work with on-chip RAM. It may also be modified to use external memory if so desired. To use external memory, modify RAM_DISK_S and RAM_DISK_E to hold the start and end addresses of external memory in the file SysMemMap.h.

7. Limitations

In the sample software since CPU-Rewrite is not implemented, data stored on the chip will be lost on power recycling. In order to prevent this, CPU-rewrite will have to be implemented so that data can be written to the ROM as opposed to the on chip RAM. Implementing this functionality will also overcome the storage area limitation (6.6 Kb) in the current design.

8. Data Sheet

1) H8SX/1664 group manual. Document number: REJ09B0294-0100

(Use the latest version on the home page: <http://www.renesas.com>)

9. References

1. H8SX/1664 group manual. Document number: REJ09B0294-0100
2. Universal Serial Bus Specification Revision 2.0
3. Universal Serial Bus Mass Storage Class (Bulk Only Transport) Revision 1.0
4. The RSK H8SX/1664 User Manual
5. "USB Complete: Everything You Need to Develop Custom USB Peripherals" by Jan Axelson.
6. [Jan Axelson's USB Mass Storage page](#)

Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (<http://www.renesas.com>).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.