R01AN2201EJ0100
Rev. 1.00
August 20, 2014

SH7216 Group

USB Function Module: USB Communication Class Application Note

Introduction

This document explains how to implement the system applied to the USB communication class as an application example for the SH7216 Group's USB function module. This document and the sample program described are examples of the USB function module, and are therefore not guaranteed by Renesas.

Target Device

This example supports the following device.

- SH7216 Group

Contents

1. Overview

This document explains how to use the SH7216 USB function module, and how to implement the system applied to the USB communication class as an application example for the USB function module.

1.1 Functions Used

- Interrupt controller (INTC)
- Pin function controller (PFC)
- USB function module (USB)
- Serial communication interface (SCI)

1.2 Applicable Conditions

MCU	SH7216
Operating frequency	Internal clock: 200 MHz Bus clock: 50 MHz Peripheral clock: 50 MHz MTU2S clock: 100 MHz AD clock: 50 MHz
Integrated development environment	Renesas Electronics High-performance Embedded Workshop, Ver. 4.07.00
C compiler	Renesas Electronics SuperH RISC engine Family C/C++ Compiler Package, Ver. 9.03.00, Release 02
Compile options	High-performance Embedded Workshop default settings (-cpu=sh2afpu -fpu=single - include="$(WORKSPDIR)\inc","$(WORKSPDIR)\src\inc" -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -gbr=auto -fpscr=safe -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 –nologo)

1.3 Related documents

The following application note is related to this application note. Refer to it as necessary in conjunction with this application note.

- SH7216 Group USB Function Module: USB Mass Storage Class Application Note (REJ06B0897)
- SH7216 Group USB Function Module: USB HID Class Application Note (REJ06B0898)
- SH7216 Group Protocol Conversion between Ethernet and USB Application Note (R01AN0066EJ)
- SH7216 Group USB Multifunction Operation of USB Function Module Application Note (R01AN0294EJ)
- SH7216 Group Using USB to Reprogram Flash Memory in User Program Mode Application Note (R01AN0316EJ)
- SH7216 Group Peripheral LibUSB Demo Application Note (R01AN0889EJ)

2. Applications

This sample program uses the USB function module to execute the control IN, control OUT, bulk IN, and bulk OUT transfers. The sample program also converts USB to serial signals and vice versa based on the USB communication class.

2.1 Features

The USB function module has an embedded USB 1.1 compliant USB Device Controller (UDC) and operates the USB protocols automatically.
Figure 2.1 is a block diagram of the USB function module.

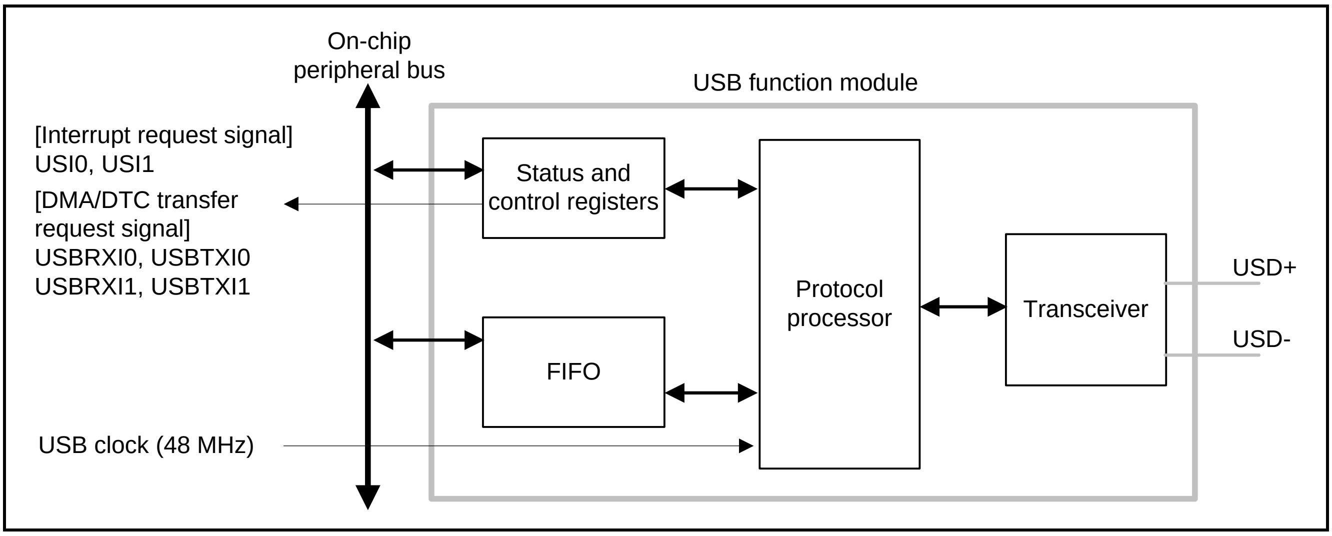

Figure 2.1 Block Diagram of USB

The features of the on-chip USB function module of the SH7216 are as follows.

- Automatic processing of USB protocol
- Automatic processing of USB standard commands for endpoint 0 (Some commands need to be processed through the software.)
- Transfer speed: Full speed
- Interrupt requests: Generation of interrupt signals needed for USB transmission and reception
- Clock: External input clock generated by USB oscillator (48 MHz)
- Low-power mode
- Integrated bus transceiver
- Endpoint configurations: There are ten mounted endpoints for each transfer mode, and transfers data in the control transfer, bulk OUT transfer, bulk IN transfer, and interrupt transfer to the USB host. Table 2.1 shows those configurations.

Table 2.1 Endpoint Configurations

Endpoint	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	—

2.2 USB Communication via the USB Function Module

As an example of using the USB function module for USB communication, the sample program implements the USB communication features listed in the table below.

Table 2.2 USB Communication Functions

USB Communication Features	Description
Detect a connection to the USB host	The port pulls up the D+ pin to detect a connection.
Control transfer	Decodes requests, processes the Data stage and Status stage of the USB request transmitted from the USB host in the control transfer.
Bulk IN/OUT transfers	Executes bulk IN/OUT transfers.

2.3 Detecting a Connection to the USB Host

A connection to the USB host is detected using the cable connect interrupt (the BRST bit in the USBIFR0 register). The cable connect interrupt occurs when a USB device cable is connected to the USB host. After the user configures the MCU, the sample program pulls up the USB data bus D+ pin using the general output port. The USB host recognizes that a USB device is connected by the pull-up. Figure 2.2 shows an operation flow of the sample program.

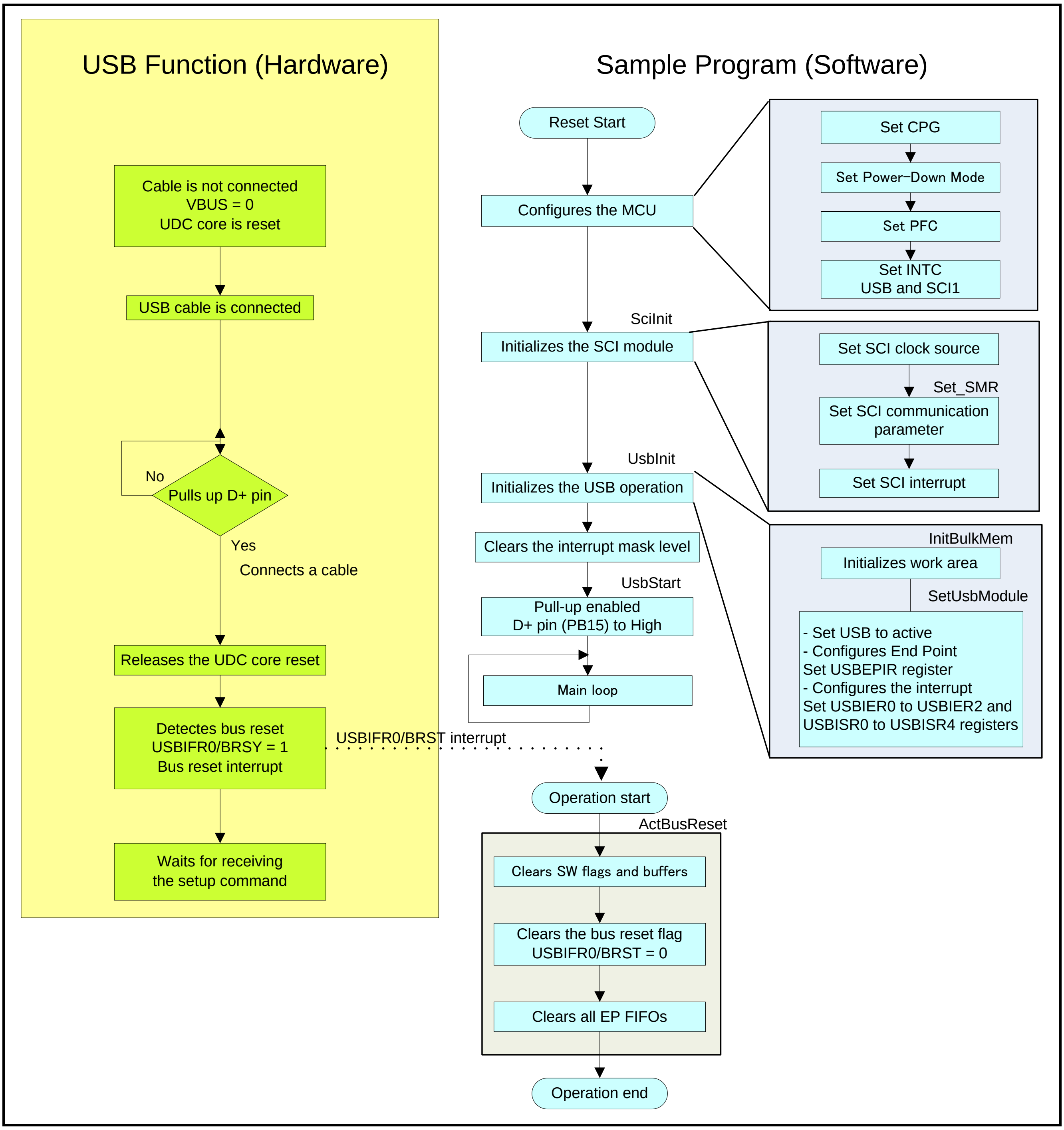

Figure 2.2 Detecting a Connection to the USB Host

2.4 Control Transfer

A control transfer is a USB transfer that uses the endpoint 0 as the default pipe and must be supported by all USB devices. The USB host issues a USB standard request to the USB device and configures the device. A control transfer can be used to issue a class- or vendor-specific request command.

A control transfer is composed of a Setup stage, Data stage (not in all cases), and Status stage. The Data stage consists of multiple bus transactions. Control transfers are supported via bi-directional communication flow, according to the data direction of the Data stage. A control OUT transfer is the data flow from the USB host to the USB device during the Data stage. A control IN transfer is the data flow from the USB device to the USB host during the Data stage. The Data stage is completed when the USB host transmits a token whose data direction is inverted to the previous data stages.

The Status stage is the stage that transmits the inverted token to the USB host. Figure 2.3 shows the Data direction in the Data stage and Figure 2.4 shows each the control transfer stage configuration.

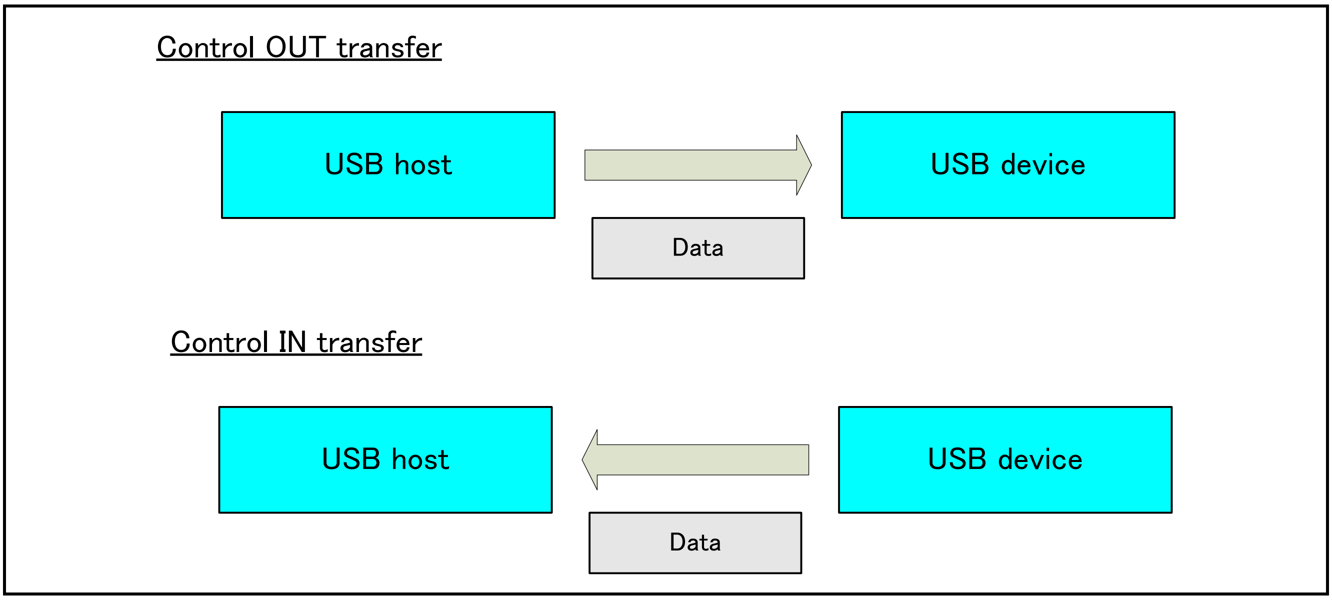

Figure 2.3 Data Direction in the Data Stage

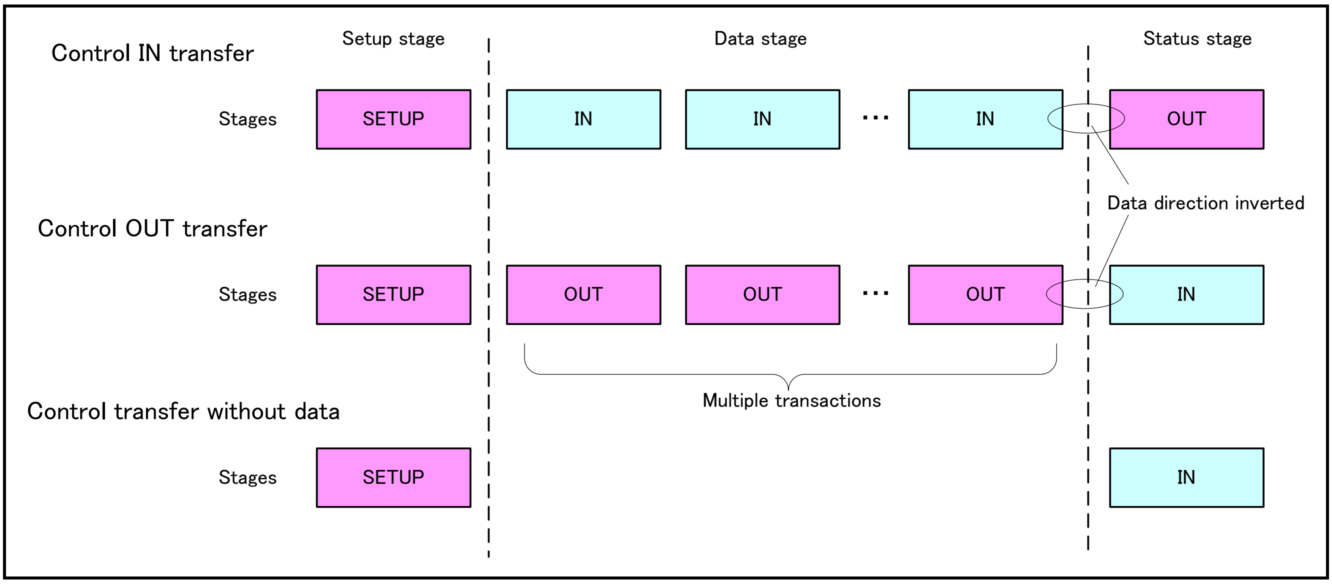

Figure 2.4 Control Transfer Stage Configuration

The USB function module decodes the request, and automatically processes the Data stage and Status stage according to the USB standard request. Some of the USB standard requests, class requests, and vendor requests should be executed by the software.
The sample program executes the Get Descriptor command, which is the USB standard request to be processed by the software and the conversion between USB and serial communication as the typical example of the USB communication class. To convert USB communication into serial communication, the sample program executes USB communication class requests. Table 2.3 lists USB commands and processing in the sample program.
When the sample program receives a USB command that it does not support, it returns a STALL handshake.

Table 2.3 USB Commands and Processing in the Sample Program

USB Command	Type	Processing in the Sample Program
Clear Feature Get Configuration Get Interface Get Status Set Address Set Configuration Set Feature Set Interface	USB standard request	The hardware decodes the command, and executes the Data stage and the Status stage automatically. The software does no operation.
Get Descriptor	USB standard request	The software decodes the command, and executes the Data stage and the Status stage.
Set Line Coding Set Control Line State Send Break Get Line Coding	USB communication class request	The software decodes the command, and executes the Data stage and the Status stage.
Other USB commands	-	The software returns a STALL handshake.

2.4.1 Setup Stage

The Setup stage is composed of one setup transaction. The USB host sends a setup token or data packet (USB command), then returns a handshake in response to the data packet (USB command) that the USB device received. Figure 2.5 shows the setup transaction sequence. The USB function module automatically executes the Setup stage, Data stage, and Status stage in response to the USB requests (with some exceptions). If the received request is not a USB standard request, the USB function module holds the received request in the EP0s data register (USBEPDR0s), and generates the setup request receive complete interrupt using the SETUPTS bit in the USBIFR1 register.
The sample program reads the USB request held in the data register (USBEPDR0s) during the interrupt, and decodes the USB request to determine how to process subsequent stages. If the decoded USB request is a request to execute the control IN transfer, the sample program writes the first data to transfer to the USB host in the EP0i FIFO and the interrupt is completed. Figure 2.6 shows an operation flow of the sample program. The function (DecComCommand) processes USB communication class requests in the sample program.

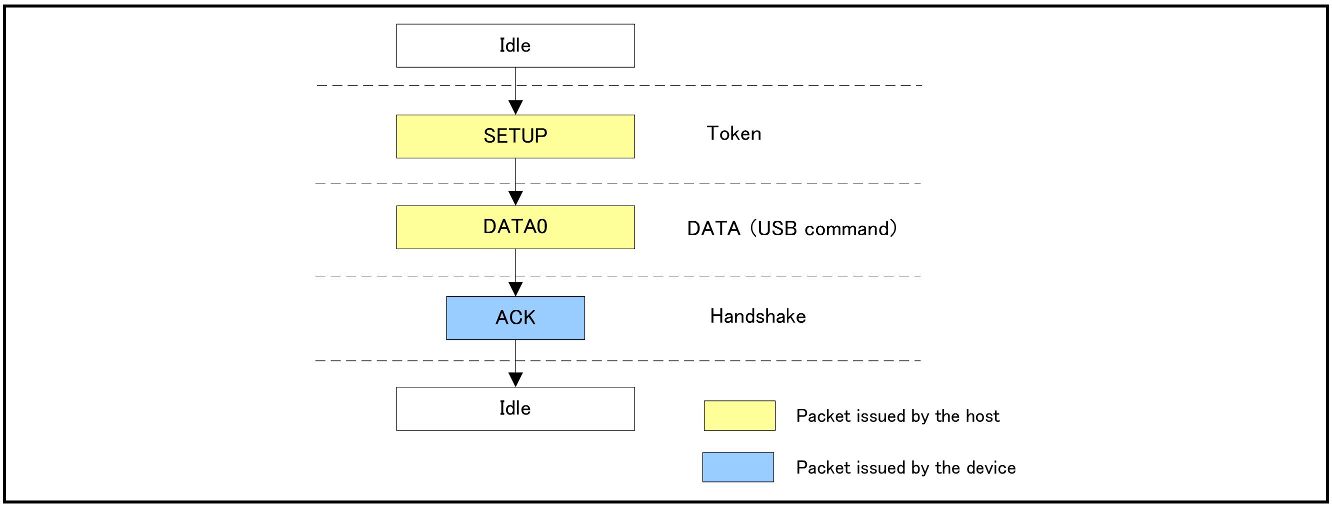

Figure 2.5 Setup Transaction Sequence

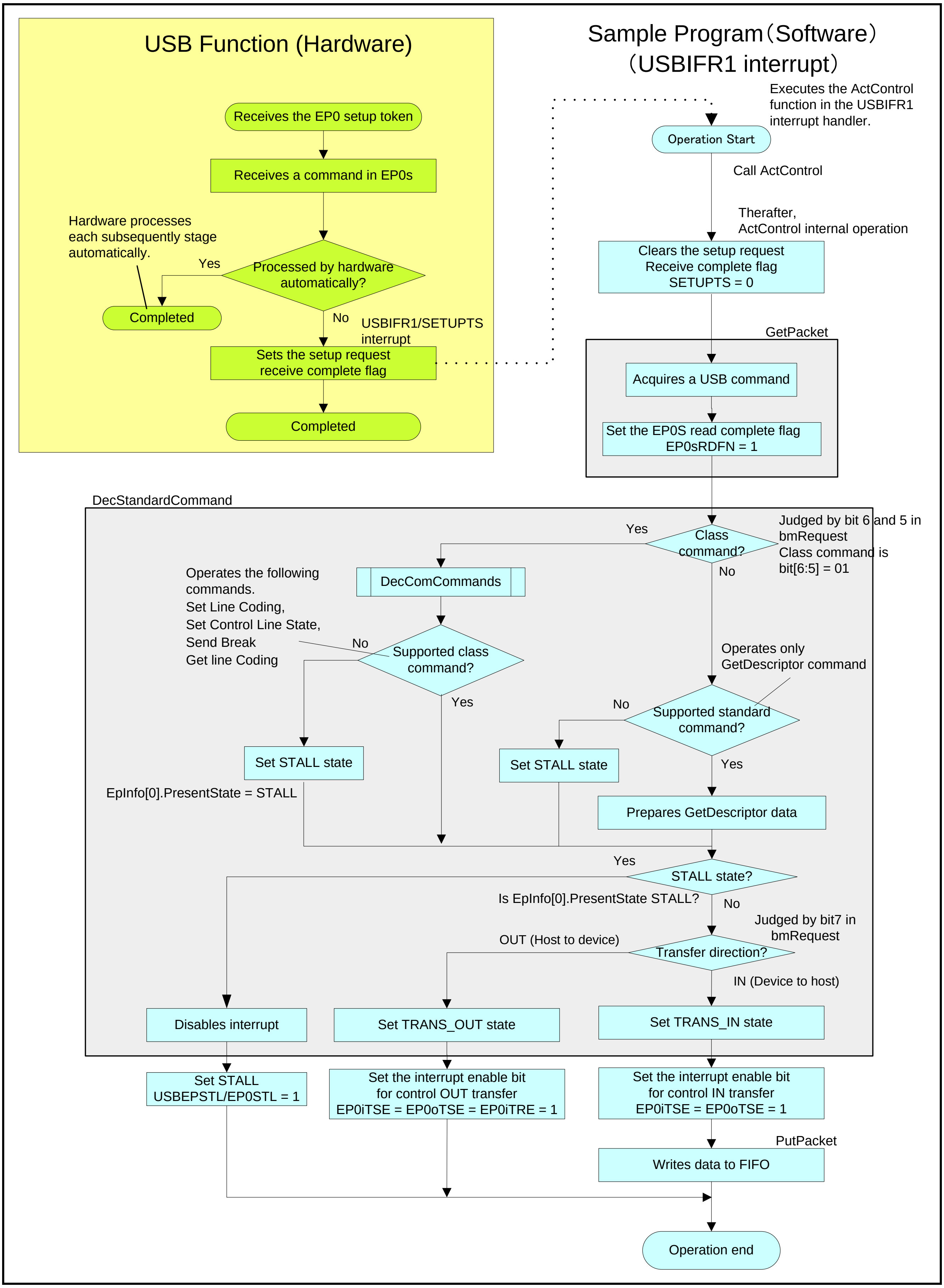

Figure 2.6 Setup Stage Operation Flow

2.4.2 Data Stage

The Data stage of the control IN transfer is composed of single or multiple data transactions. The Data stage of the control IN transfer is composed of one data IN transaction. This is called “Data IN stage processing”. The Data stage of the control OUT transfer is composed of one data OUT transaction. This is called “Data OUT stage processing”.

(1) Data IN Stage

First, the USB host sends an IN token. When the USB device receives the IN token, it sends a data packet to the USB host and waits for an ACK handshake from the USB host. If the USB device cannot send a data packet when it receives the IN token, it returns a NAK handshake to the USB host. Figure 2.7 shows the data IN transaction sequence.

When the USB function module receives an IN token without valid data in the EP0iFIFO, it automatically returns a NAK handshake to the USB host. When the module receives an IN token with valid data in the EP0iFIFO, it sends the data in the EP0iFIFO to the USB host and waits for an ACK handshake from the USB host. When the USB function module receives an ACK handshake, it generates the data transmit complete interrupt using the EP0iTS bit in the USBIFR0 register. On the contrary, when the module receives an OUT token that indicates the Data IN stage has been completed, it generates the data receive complete interrupt using the EP0oTS bit in the USBIFR0 register.

The sample program identifies the type of the interrupt while it is being processed. When it is the data receive complete interrupt (the EP0oTS bit in the USBIFR0 register), the sample program advances to the Status stage. When it is the data transmit complete interrupt (the EP0iTS bit in the USBIFR0 register), data that should be sent to the USB host was written in the EP0iFIFO, and the sample program waits for the next interrupt. Figure 2.8 shows an operation flow of the Data IN stage by the sample program.

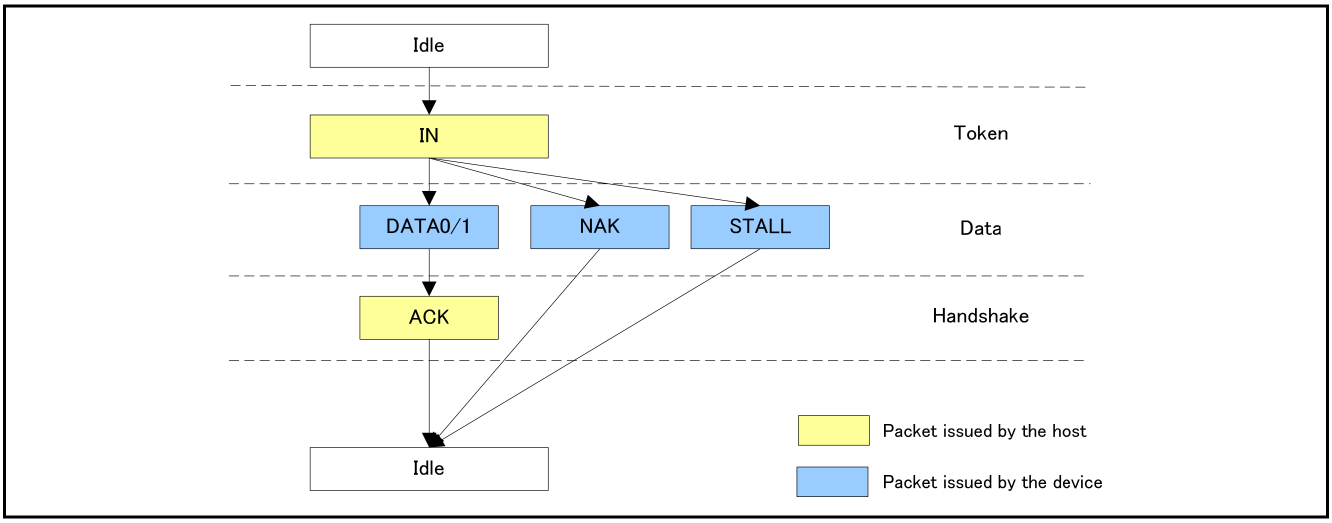

Figure 2.7 Data IN Transaction Sequence

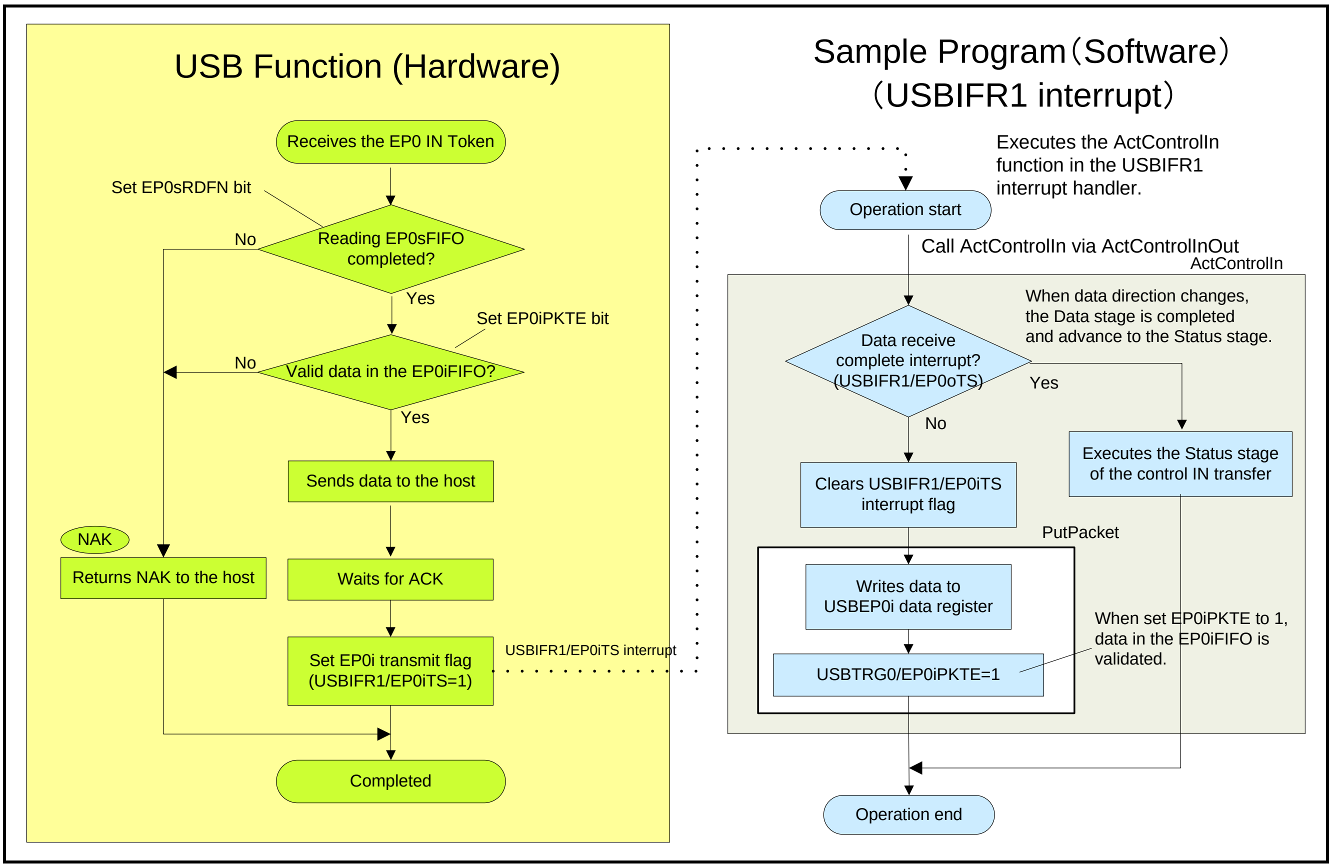

Figure 2.8 Data IN Stage Operation Flow

(2) Data OUT Stage
The USB host sends an OUT token and a data packet. The USB device receives the OUT token first, then receives the data packet, and then returns an ACK handshake. When the USB device cannot receive the data packet after it receives the OUT token, it ignores subsequent data packets, and returns a NAK handshake. When the USB host receives the NAK handshake, it tries to resend the OUT token and data packet. Figure 2.9 shows the Data OUT transaction sequence.
When the USB function module cannot receive data packets, when it receives the OUT token, it automatically discards the subsequent data packet and returns a NAK handshake to the USB host. When it receives the OUT token, when it can accept data, it holds the data packet from the USB host in the EP0oFIFO and returns an ACK handshake to the USB host. After the USB function module transmits an ACK handshake, it generates the data receive complete interrupt using the EP0oTS bit in the USBIFR1 register. When the function module receives the IN token that indicates the Data OUT stage is completed, it generates the IN token receive interrupt using the EP0iTR bit in the USBIFR1 register.
The sample program identifies the type of the interrupt while it is being processed. When it is not the data receive complete interrupt (the EP0oTS bit in the USBIFR1 register), the sample program advances to the Status stage. When it is the data receive complete interrupt (the EP0oTS bit in the USBIFR1 register), the sample program reads data in the EP0oFIFO, sets the EP0oFIFO read complete bit, and waits for the next interrupt to be generated. Figure 2.10 shows an operation flow of Data OUT stage by the sample program.

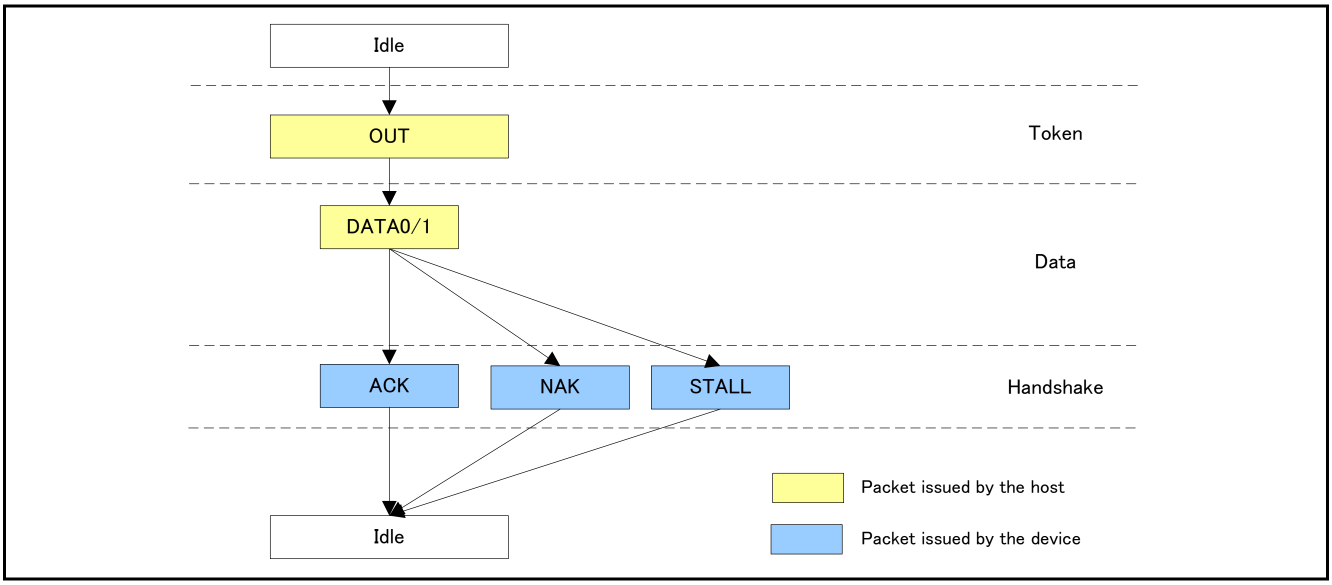

Figure 2.9 Data OUT Transaction Sequence

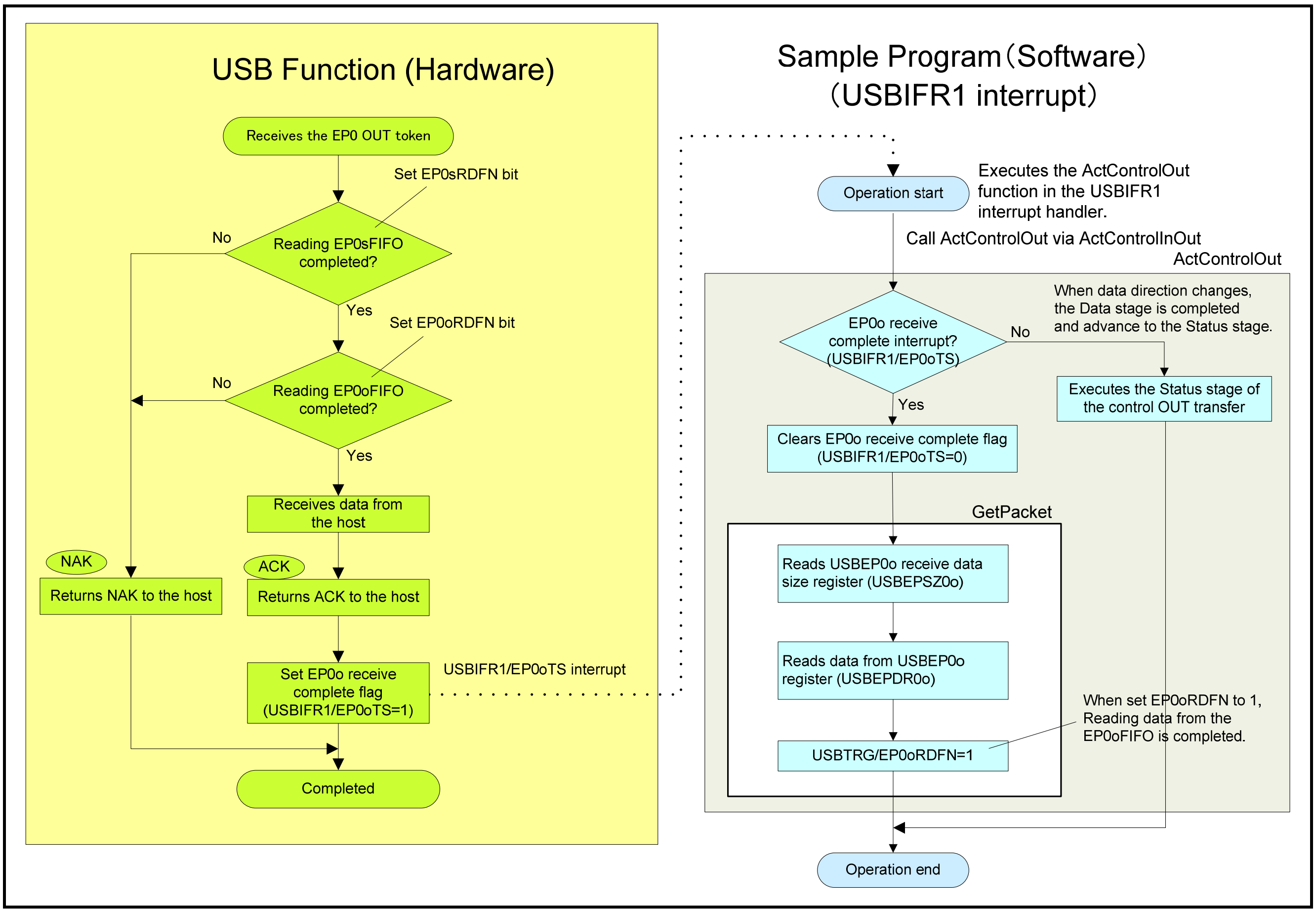

Figure 2.10 Data OUT Stage Operation Flow

2.4.3 Status Stage

The direction of the data transaction in the Status stage differs from that of the Data stage; the data OUT transaction is executed during the Status stage of the control IN transfer, and the data OUT transaction is executed during the Status stage of the control OUT transfer.

(1) Status Stage of the Control IN Transfer
The USB host sends an OUT token and a 0-byte data packet. The USB device first receives the OUT token and then receives the 0-byte data packet. Then, it returns an ACK handshake to the USB host.
The USB function module receives the OUT token and the 0-byte data packet, and automatically sends an ACK handshake to the USB host. Then, the USB function module generates the data receive complete interrupt using the EP0oTS bit in the USBIFR1 register.
The sample program set the EP0oFIFO read complete bit (USBTRG0/EP0oRDFN) during an interrupt to wait for the next interrupt. Figure 2.11 shows an operation flow of the Status stage (control IN transfer) by the sample program.

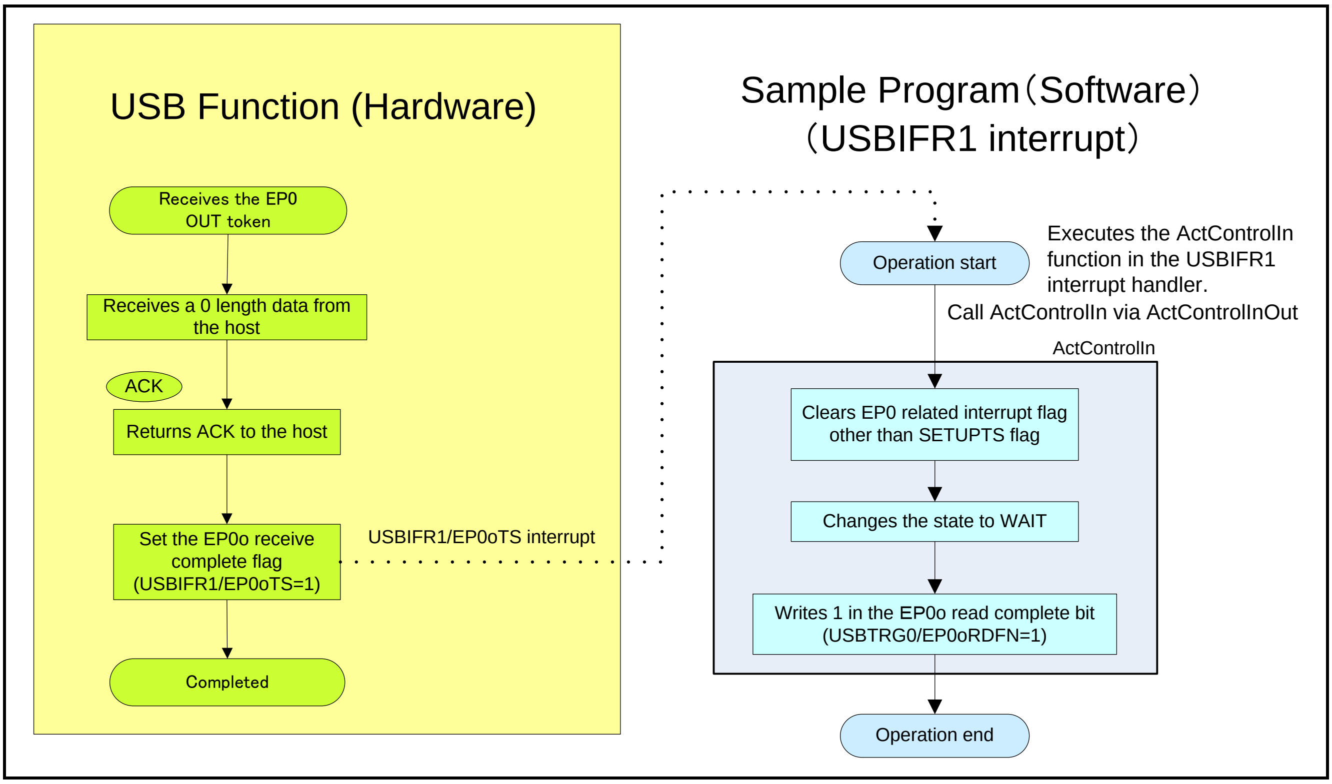

Figure 2.11 Status Stage (Control IN Transfer) Operation Flow

(2) Status Stage of the Control OUT Transfer
The USB host sends an IN token. The USB device sends a 0-byte data packet to the USB host after it receives the IN token. Then, the USB device waits for an ACK handshake from the USB host.
When the USB function module receives the IN token, it generates the IN token receive interrupt using the EP0iTR bit in the USBIFR1 register. When the USB function module receives an IN token, when there is no valid 0-byte data packet in the EP0iFIFO, it automatically returns a NAK handshake to the USB host. When the USB function module has a valid 0-byte data packet in the EP0iFIFO when it receives the IN token, it sends a 0-byte data packet to the USB host and waits for an ACK handshake from the USB host. When the USB function module receives the ACK handshake, it generates the data transmit complete interrupt using the EP0iTS in the USBIFR1 register.
The sample program identifies the type of the interrupt while it is being processed. When it is the data transmit complete interrupt generated by the EP0iTS in the USBIFR1 register, the sample program clears the interrupt to complete the control transfer. When it is the IN token receive interrupt generated by the EP0iTR bit in the USBIFR1 register, the sample program enables 0-byte data in the EP0iFIFO by setting the EP0iPKTE bit in the USBTRG1 register to 1, and waits for the next interrupt to be generated. Figure 2.12 shows an operation flow of the Status stage (control OUT transfer) by the sample program. When the command is a USB communication class command (Set Line Coding), the sample program calls a SciInit function to process the command.

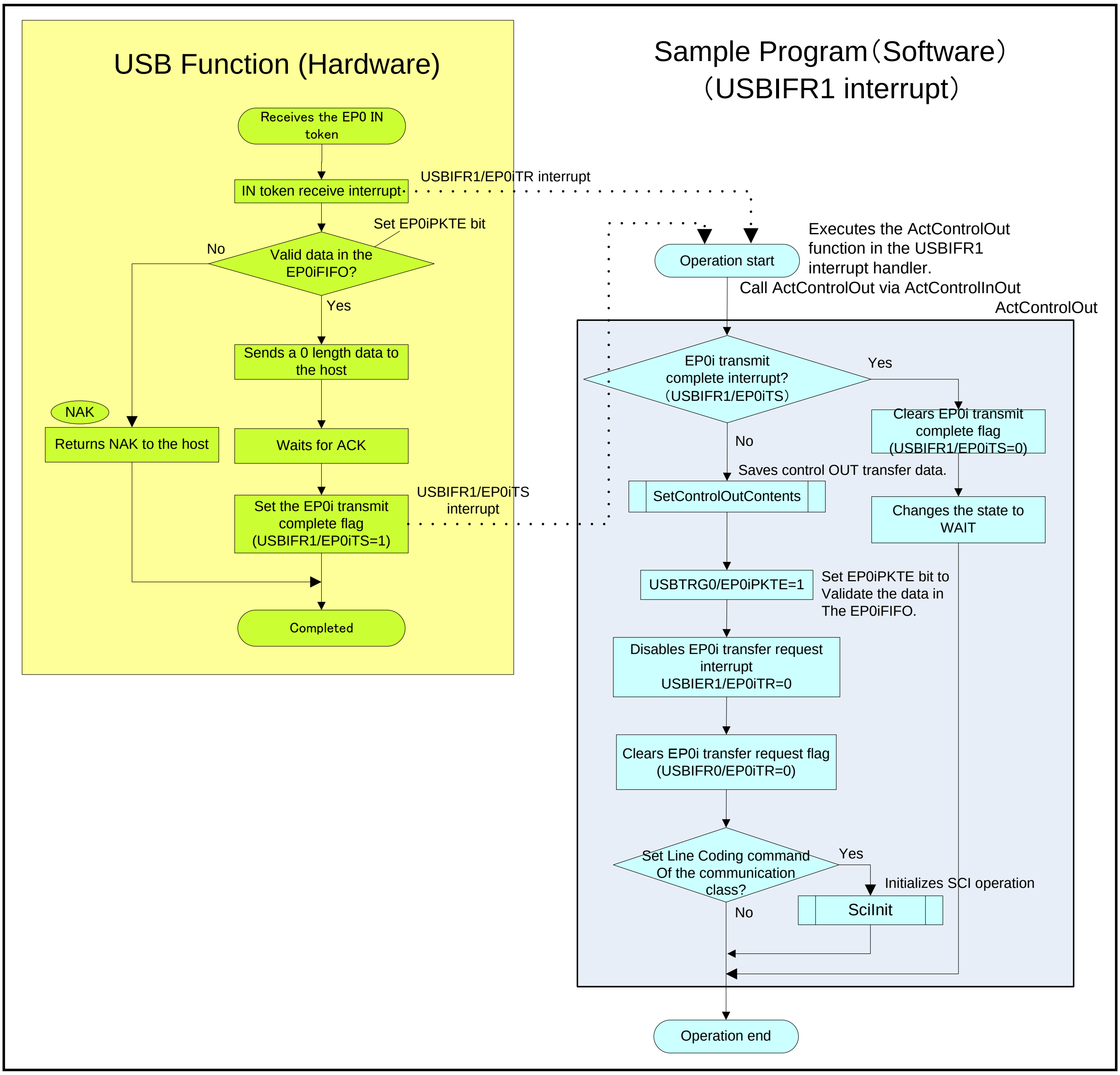

Figure 2.12 Status Stage (Control OUT Transfer) Operation Flow

2.5 Bulk Transfer

A bulk transfer is used to communicate large amounts of data between the USB host and the USB device. Bulk transfers are supported via bi-directional communication flow for the data direction. A bulk OUT transfer is the data flow from the USB host to the USB device, and a bulk IN transfer is the data flow from the USB device to the USB host. Figure 2.13 shows the data directions of the bulk IN and bulk OUT transfers.

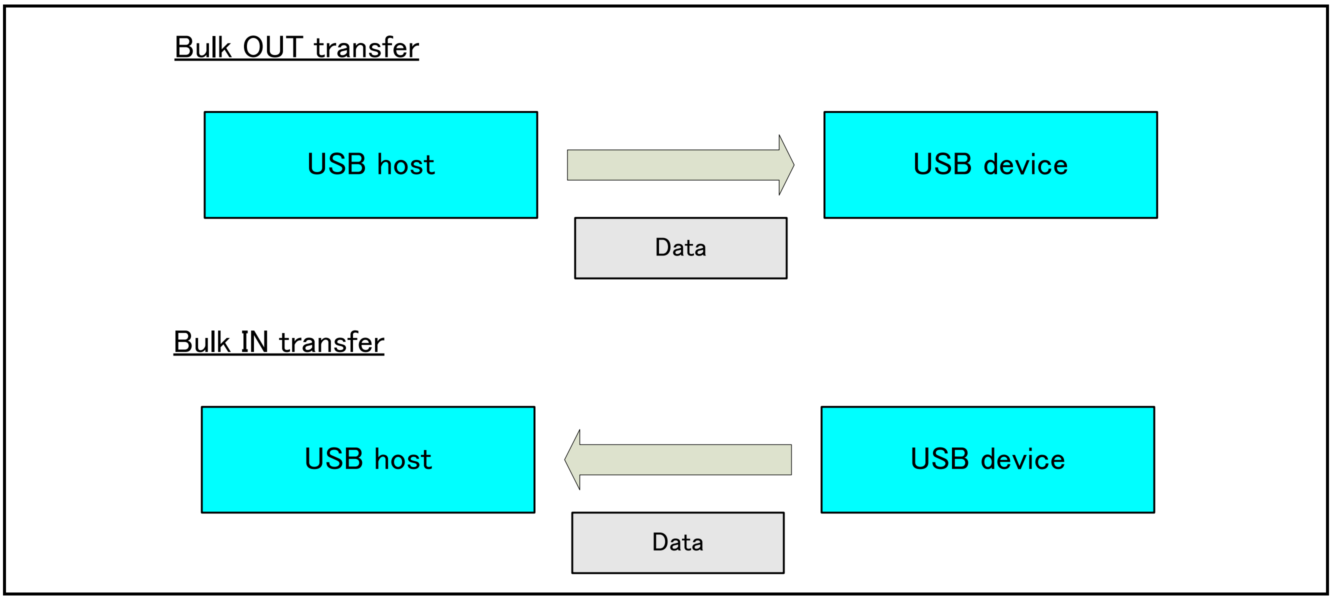

Figure 2.13 Data Direction in the Bulk Transfer

2.5.1 Bulk OUT Transfer

A bulk OUT transfer is composed of single or multiple OUT transactions. The USB host sends an OUT token and a data packet in the OUT transaction. The USB device receives the out token first, then receives the data packet, and then returns an ACK handshake. When the USB device receives the OUT token when it is unable to receive data packets, it ignores subsequent data packets and returns a NAK handshake. When the USB host receives a NAK handshake, it attempts to resend the OUT token and data packet to the USB device. Figure 2.14 shows the OUT transaction sequence.
When the USB function module receives an OUT token when it is unable to receive data packets, it automatically discards subsequent data packets and returns a NAK handshake to the USB host. When the USB function module receives an OUT token when it can accept data packets, it holds the data from the USB host in the EP1FIFO and returns an ACK handshake to the USB host. Then, the USB function module generates the data receive complete interrupt using the EP1FULL bit in the USBIFR1 register.
The sample program reads the data in the EP1FIFO and stores it in the bulk OUT transfer RAM during the interrupt, set the EP1 read complete flag (by setting the EP1RDFN bit in the USBTRG register to 1), and waits for the next interrupt to be generated. When the interrupt is generated, if there is no area in the EP1FIFO to store data, the sample program disables the EP1FULL interrupt, and the interrupt process is completed while it is in a suspended state. When serial communication is used to create an area in the bulk OUT transfer RAM, enabling the EP1FULL interrupt causes the suspended EP1FULL interrupt to be generated, and moves the data in the EP1FIFO to the bulk OUT transfer RAM in the interrupt handler, and sets the EP1 read complete flag to wait for the next interrupt to be generated.

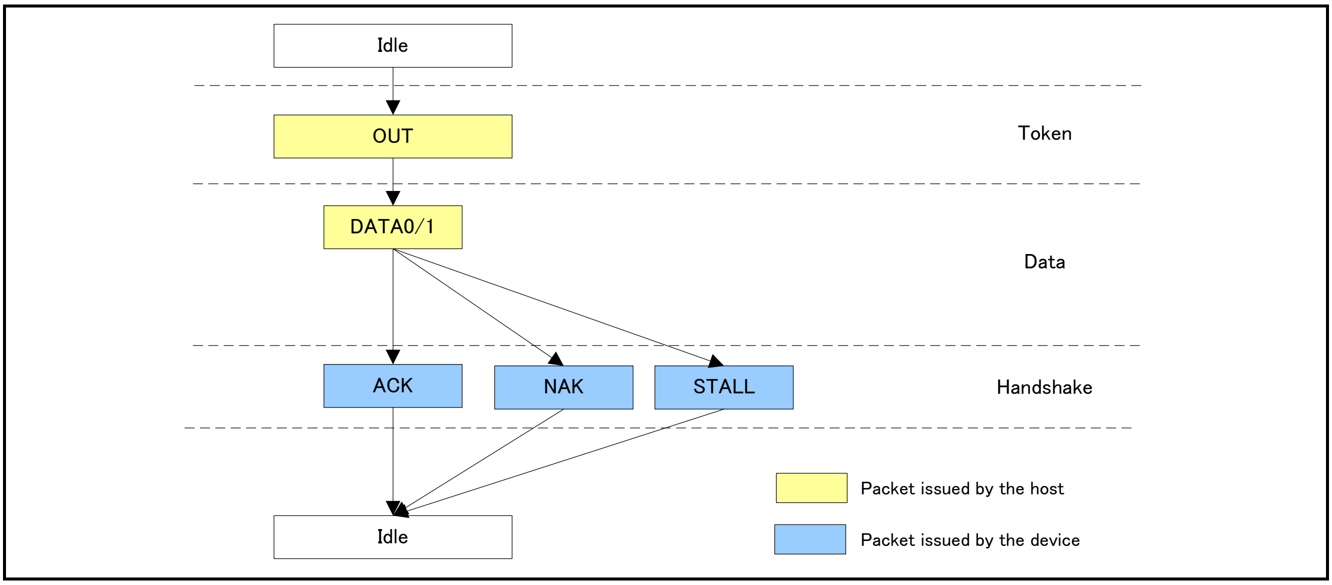

Figure 2.14 Bulk OUT Transaction Sequence

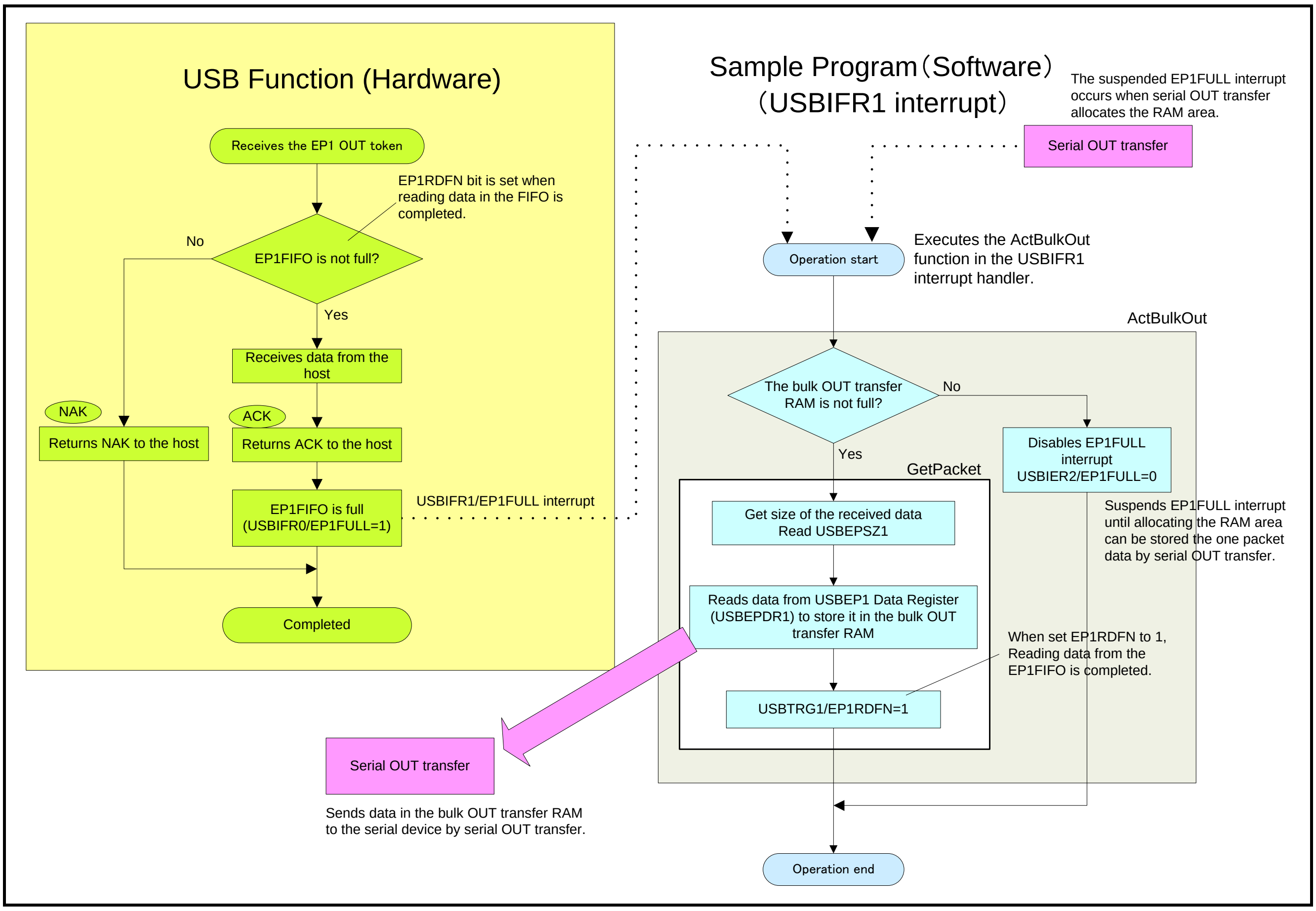

Figure 2.15 Bulk OUT Transfer Operation Flow

2.5.2 Bulk IN Transfer

A bulk IN transfer consists of single or multiple transactions. The USB host sends an IN token in the IN transaction. When the USB device receives the IN token, it sends a data packet to the USB host and waits for an ACK handshake from the USB host. When the USB device receives the IN token when it is unable to send data packets, it returns a NAK handshake to the USB host. Figure 2.16 shows the IN transaction sequence.

When the USB function module receives an IN token when there is no valid data in the EP2FIFO, it automatically returns a NAK handshake to the USB host. The USB function module determines if there is any valid data in the EP2FIFO using the EP2PKTE bit in the USBTRG register. When the module receives the IN token when there is valid data in the EP2FIFO, the USB function module sends the data in the EP2FIFO to the USB host and waits for an ACK handshake from the USB host. When the ACK handshake is received, the USB function module sets the data transmit complete flag using the EP2EMPTY bit in the USBIFR2 register.

The sample program periodically transfers data in bulk transfer mode within the main loop. When the serial communication generates data in the bulk IN transfer RAM that should be transmitted to the USB host, the sample program confirms if the EP2FIFO is empty. When the EP2FIFO is empty, the sample program writes the data in the EP2FIFO, sets the PE2PKTE bit to 1, and enables the data in the EP2FIFO. When the EP2FIFO is not empty, the sample program is idle and waits for the USB function module to finish transferring data to the USB host.

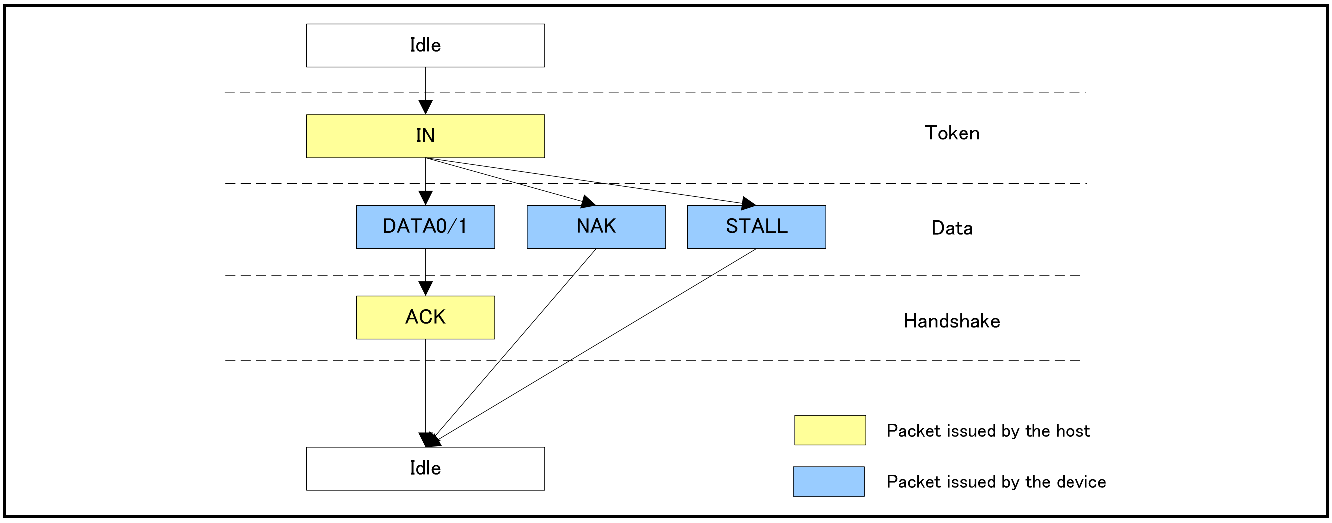

Figure 2.16 Bulk IN Transaction Sequence

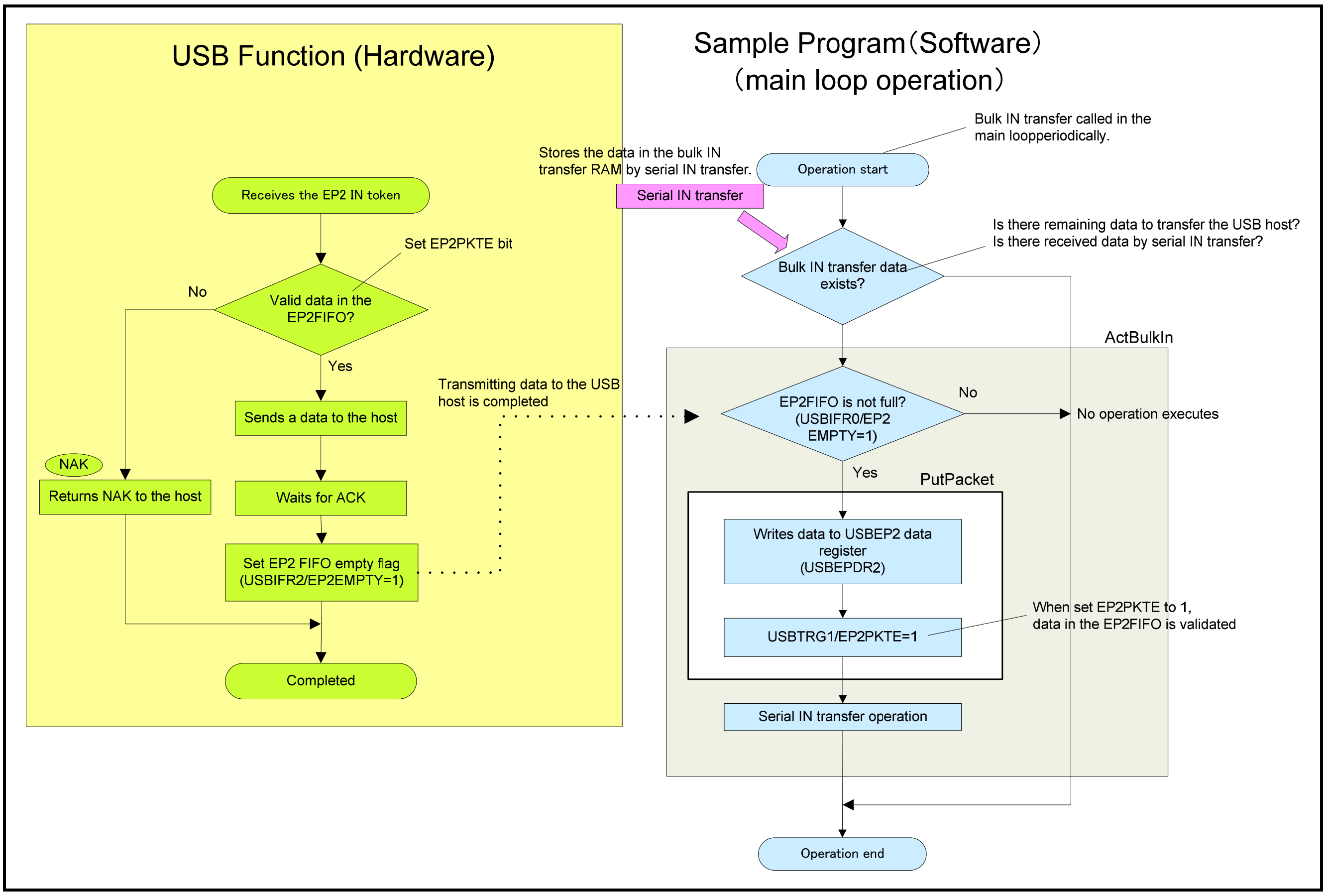

Figure 2.17 Bulk IN Transfer Operation Flow

3. System Example of the USB Communication Class

USB to serial conversion system is implemented in this sample program as the USB communication class typical example.

3.1 Overview

The sample program converts USB to serial using the USB function module and the serial communication interface (SCI) embedded in the SH7216 MCU. To transfer keyboard-input characters, text files, or binary files, activate the terminal software both on the USB host (PC in this case), and on the serial device. For example, some characters are input using the keyboard of the USB host, and the characters are transferred to the serially connected device. On the other hand, some characters are input using keyboard of the serially connected device for transfer to the USB host.
A system configuration of the USB to serial conversion is shown in Figure 3.1, and Table 3.1 lists its specifications.

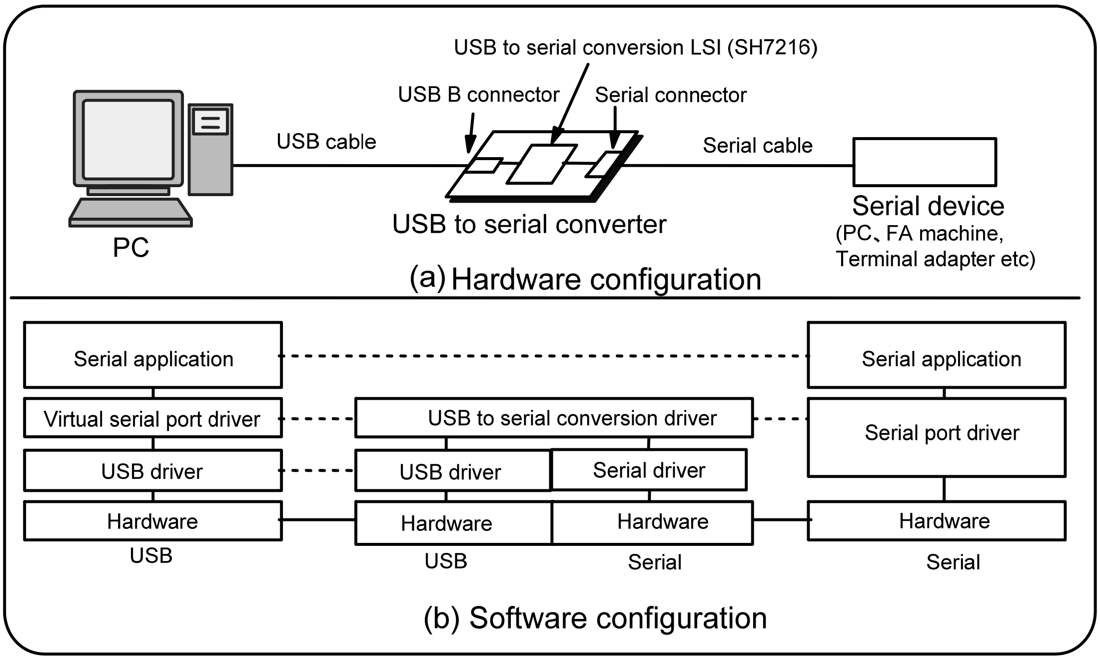

Figure 3.1 System Configuration

Table 3.1 System Specification

Features	Description
Detect a connection to the USB host	The port pulls up the D+ pin to detected by the USB host.
Control transfer (USB standard request)	(1) Decode requests, processes the Data stage and Status stage of the USB request transmitted from the USB host in the control transfer. (2) Sends the descriptor information to Get Descriptor command to connect with the USB host as a USB communication class. (3) Descriptor samples are described in the SetUsbInfo.h file and those data are send to the USB host.
Control transfer (USB communication class request)	Supports the following USB communication class commands: Get Line Coding, Set Line Coding, Set Control Line State, and Send Break.

Features	Description
Bulk IN/OUT transfer	Executes the bulk IN/OUT transfer.
USB to serial data conversion	(1) Sends the received data from the USB host in bulk OUT transfer to the serial device by serial OUT transfer. (2) Sends the received data from the serial device by serial IN transfer to the USB host in bulk IN transfer.

Note:
Table 3.2 lists the default value of the vendor ID and product ID in the device descriptor sample. These values MUST be changed when applied to the user system.

Table 3.2 Vendor ID and Product ID Sample

ID	Value	Description
Vendor ID	0x045B	Renesas Electronics
Product ID	0x0020	SH7216 USB communication class driver

3.2 Operation Flow

The sample program enters the main loop after initial settings are completed.
When the sample program receives a data packet from the serial device in serial IN transfer, it stores the received packet in the bulk IN transfer RAM. The stored data is transmitted to the USB host in a bulk IN transfer. For details on bulk IN transfer, refer to 2.5.2. For details on serial IN transfer, refer to 3.3.2.
When the sample program receives a data packet from the USB host in a bulk OUT transfer, it stores the received data packet in the bulk OUT transfer RAM. The stored data is transmitted to the serial device in the serial OUT transfer. For details on the bulk OUT transfer, refer to 2.5.2. For details on serial OUT transfer, refer to 3.3.1.
Figure 3.2 shows the data flow and Figure 3.3 shows an operation flow.

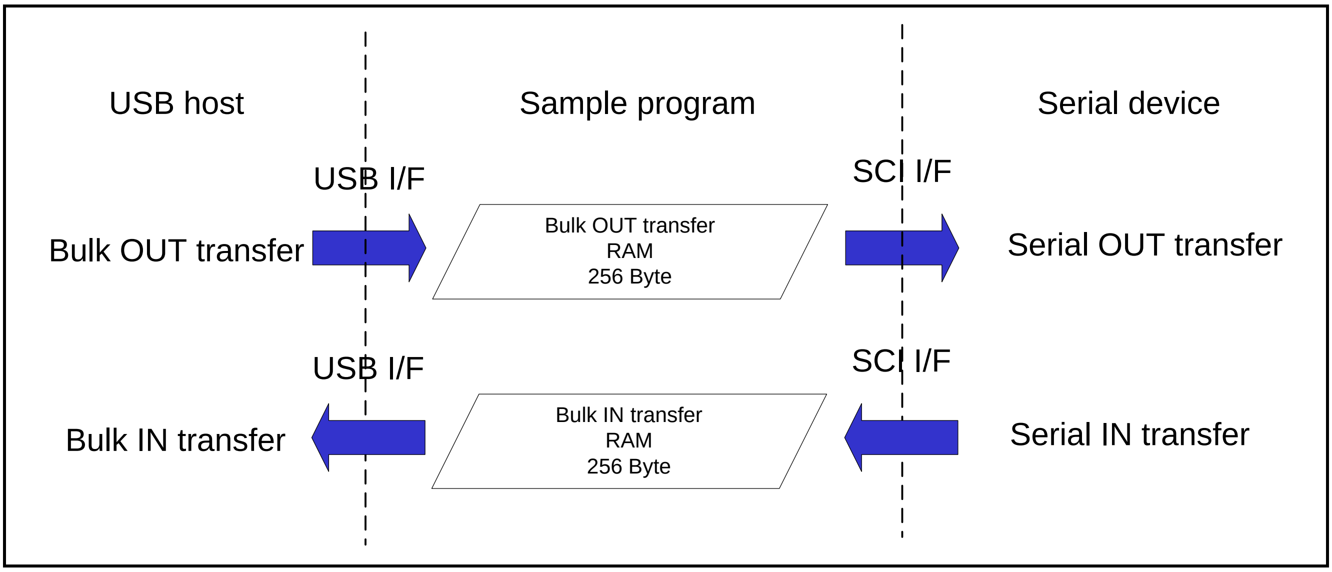

Figure 3.2 Data Flow

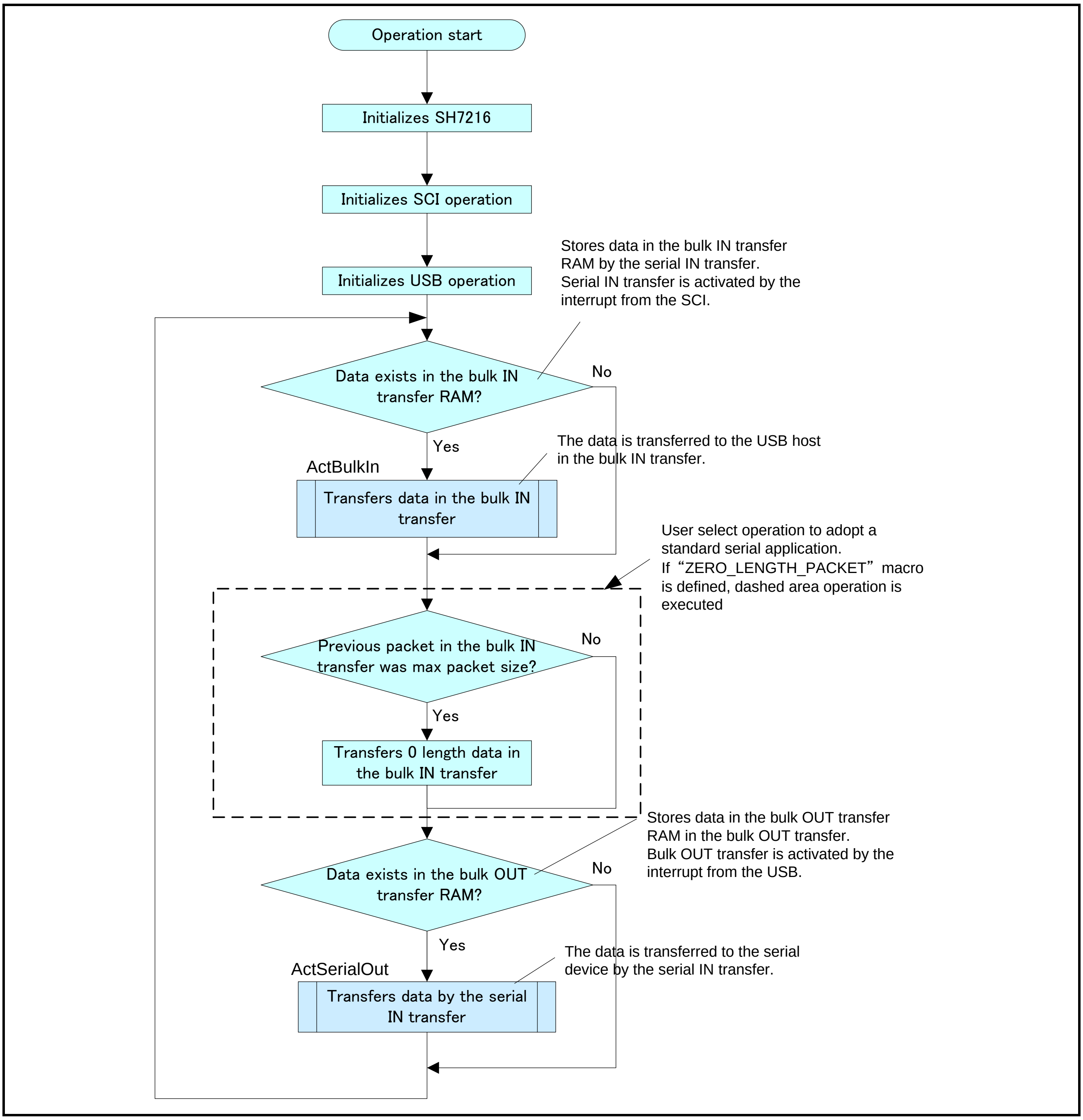

Figure 3.3 Operation Flow

3.3 Serial Communication

This sample program uses a SCI module for serial communication. The serial OUT transfer is executed only when there is data in the bulk OUT transfer RAM, and the serial IN transfer is executed only by the serial receive interrupt.

3.3.1 Serial OUT Transfer

This sample program uses the ActSerialOut function to execute serial OUT transfer. Data in the bulk OUT transfer RAM is serially transmitted in serial OUT transfer. When the bulk OUT transfer RAM can store the data by the serial OUT transfer, the sample program enables the EP1FULL interrupt. When the EP1FULL interrupt is enabled, the suspended EP1FULL interrupt occurs to transfer data in the bulk OUT transfer. Figure 3.4 shows an operation flow of the serial OUT transfer.

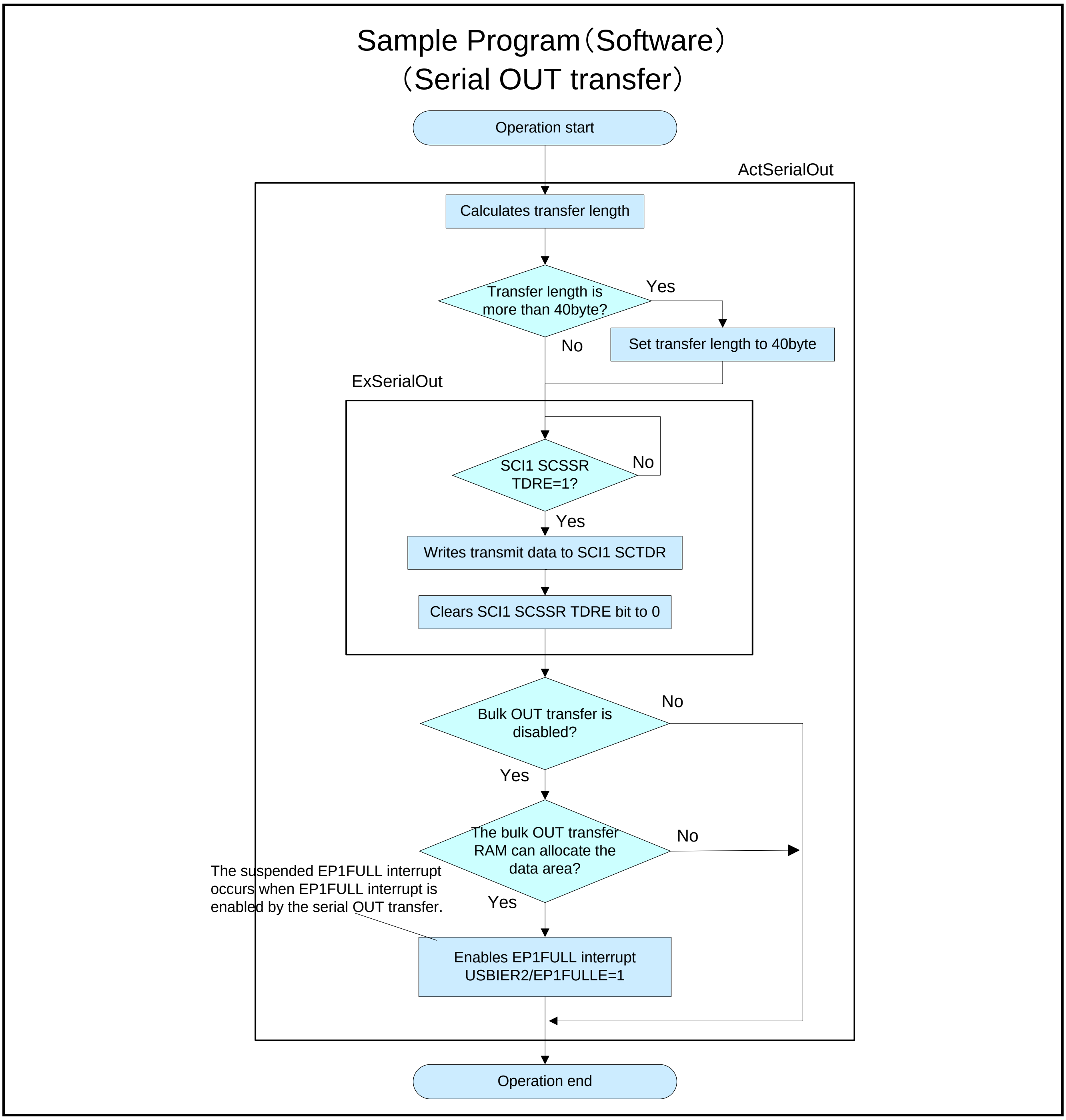

Figure 3.4 Serial OUT Transfer Operation Flow

3.3.2 Serial IN Transfer

Serial IN communication is activated by the serial receive interrupt and executed by the ActSerialIn function. The sample program stores serially-received data in the RAM for the bulk IN transfer. When the RAM is full, the sample program transmits Xoff in serial communication to disable the serial IN communication. Figure 3.5 shows an operation flow of the Serial IN Communication.

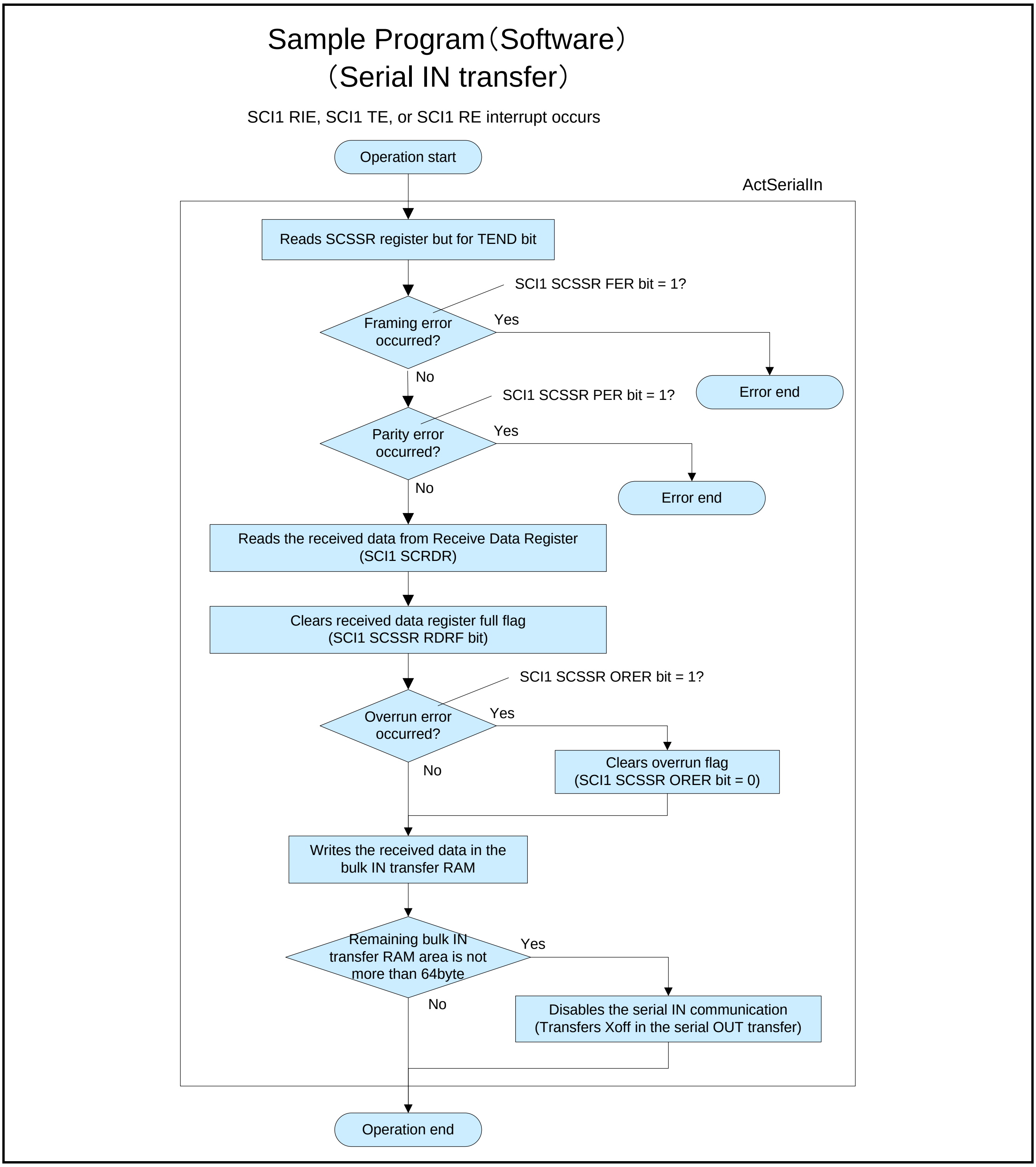

Figure 3.5 Serial IN Transfer Operation Flow

3.4 Environment Setting

Use a USB cable to connect the SH7216 CPU board (R0K572167C001BR) to the PC1. Then, use an RS-232C serial cable to connect the SH7216 CPU board to the PC2 (cross-connect). PC1 behaves as the USB host and PC2 behaves as the serial device. Figure 3.6 shows the USB to serial conversion environment.

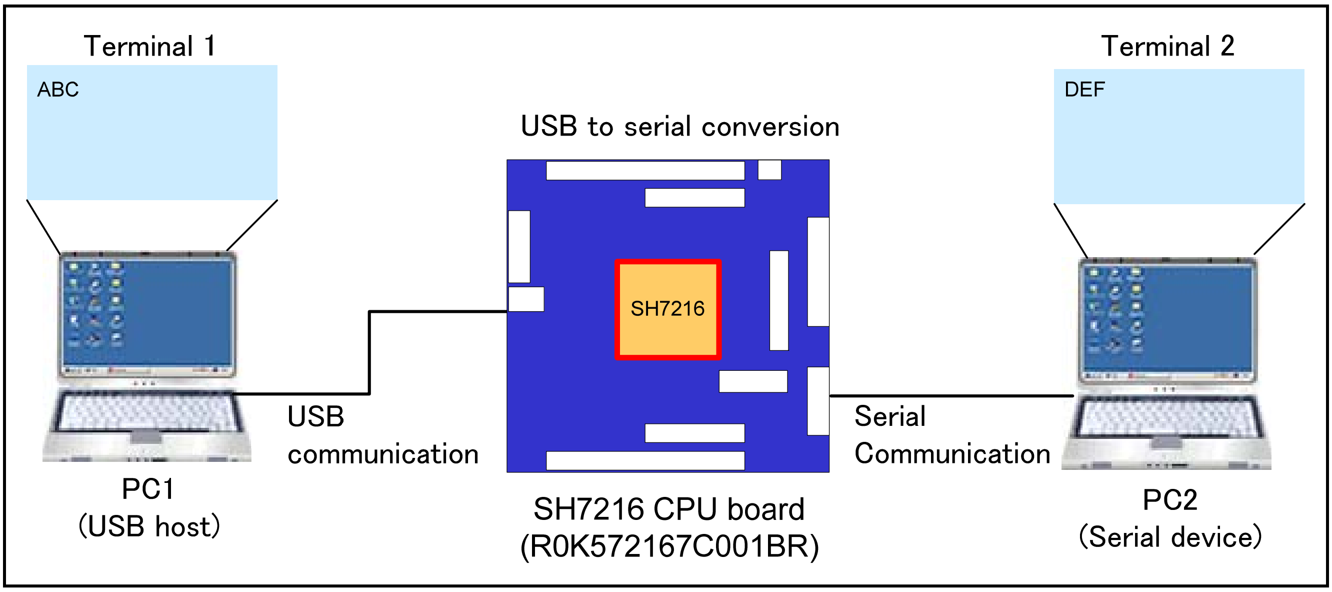

Figure 3.6 USB to Serial Conversion Environment

3.4.1 INF Files

After the sample program is loaded and written to the SH7216 CPU board and is connected to the PC1 with a USB cable for the first time, the device driver needs to be installed on the USB host computer. Install the Windows® standard USB communication class device driver (usbser.sys) as the device driver.
INF file is used for the driver installation. This sample program includes following INF files in the inf directory.

(1) RN_CommClass_32.inf
Windows® 2000, Windows® XP, Windows® Vista 32bit, and Windows® 7 32bit.

(2) RN_CommClass_64.inf
Windows® Vista 64bit and Windows® 7 64bit.

Note:
VID_045B and PID_0020 MUST be changed according to the Vendor ID and Product ID set in the device descriptor. Table 3.3 shows the default values.

Table 3.3 Vendor ID and Product ID in the INF File

Value	Description
VID_045B	Indicates the Vendor ID = 0x045B.
PID_0020	Indicates the Product ID = 0x0020.

3.4.2 Parameter Setting of Serial Application

You need to set the parameter of the serial applications (terminal 1 and terminal 2) on the PC. Table 3.4 lists the settings.

Table 3.4 Parameter Setting

Items	Description
Connection	Select the port number connected to an RS-232C serial cable or a USB cable.
Bit rate (B)	115,200 bps
Data bits (D)	8
Parity (P)	None
Stop bit (S)	1
Flow control (F)	Xon/Xoff

4. Reference Documents

User's Manual: Hardware

SH7214 Group, SH7216 Group User's Manual: Hardware Rev.4.00 (R01UH0230EJ0400)

User's Manual: Software

SH-2A, SH2A-FPU User's Manual: Software Rev.3.00 (REJ09B0051)

Website and Support

Renesas Electronics website

http://www.renesas.com

Inquiries

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

REVISION HISTORY	SH7216 Group USB Function Module: USB Communication Class Application Note

Rev.	Date	Description	
		Page	Summary
1.00	Aug 20, 2014	—	First edition issued.
-	-	—	

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins

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

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

2. Processing at Power-on

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

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

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

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

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

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

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

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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

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

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

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

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141