To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

RENESAS
Renesas Technology Corp.

Cautions

Application Notes

16

USB Function Module USB Serial Conversion

Application Notes

Renesas 16-Bit Single-Chip Microcomputer

H8S/2215
HD64F2215

Renesas Electronics
www.renesas.com

Rev.1.0 2002.04

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

Preface

These application notes describe the USB serial conversion firmware which uses the USB Function Module that incorporates the H8S/2215. They are provided to be used as a reference when the user creates USB Function Module firmware.

These application notes and the described software are application examples of the USB Function Module, and their contents and operation are not guaranteed.

In addition to these application notes, the manuals listed below are also available for reference when developing applications.

[Related manuals]

- Universal Serial Bus Specification Revision 1.1
- H8S/2215 Series Hardware Manual
- H8S/2215 CPU Board (MS2215CP01-C/S) Instruction Manual
- H8S/2215 Series TFP-120 User System Interface Cable (HS2215ECN61H) Instruction Manual
- E6000 (HS2214EPI61H) Emulator User's Manual

[Caution] The sample programs described in these application notes do not include firmware related to interrupt transfer and isochronous transfer, which are USB transport types. When using either of these transfer types (see section 15 in the H8S/2215 Series Hardware Manual), the user needs to create the program for it.

Also, the hardware specifications of the H8S/2215 and H8S/2215 CPU board, which will be necessary when developing the system described above, are described in these application notes, but more detailed information is available in the H8S/2215 Series Hardware Manual and the H8S/2215 CPU Board Instruction Manual.

Contents

Section 1 Overview

1.1 Overview

These application notes describe how to use the USB Function Module that is built into the H8S/2215, and examples of firmware programs.

The features of the USB Function Module contained in the H8S/2215 are listed below.

- An internal UDC (USB Device Controller) conforming to USB 1.1
- Automatic processing of USB protocols
- Automatic processing of USB standard commands for endpoint 0 (some commands need to be processed through the firmware)
- Full-speed (12 Mbps) transfer supported
- Various interrupt signals needed for USB transmission and reception are generated
- Internal system clock (16 MHz) multiplied by three or external input clock (48 MHz) can be selected as the USB operating clock by the USB clock selector in the clock pulse generator
- Low power consumption mode provided
- An internal bus transceiver

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	64 bytes	64 bytes	—
	EP0o	Control Out	64 bytes	64 bytes	—
Endpoint (optional)	EPn	Interrupt (in)	64 bytes	64 bytes (variable)	—
Endpoint (optional)	EPn	Bulk-in	64 bytes	64 x 2 (128 bytes)	Possible
Endpoint (optional)	EPn	Bulk-out	64 bytes	64 x 2 (128 bytes)	Possible
Endpoint (optional)	EPn	Isochronous (in)	128 bytes	128 x 2 (variable)	—
Endpoint (optional)	EPn	Isochronous (out)	128 bytes	128 x 2 (variable)	—
Endpoint (optional)	EPn	Bulk-in	64 bytes	64 x 2 (128 bytes)	Possible
Endpoint (optional)	EPn	Bulk-out	64 bytes	64 x 2 (128 bytes)	Possible
Endpoint (optional)	EPn	Interrupt (in)	64 bytes	64 bytes (variable)	—

Figure 1.1 shows an example of a system configuration.

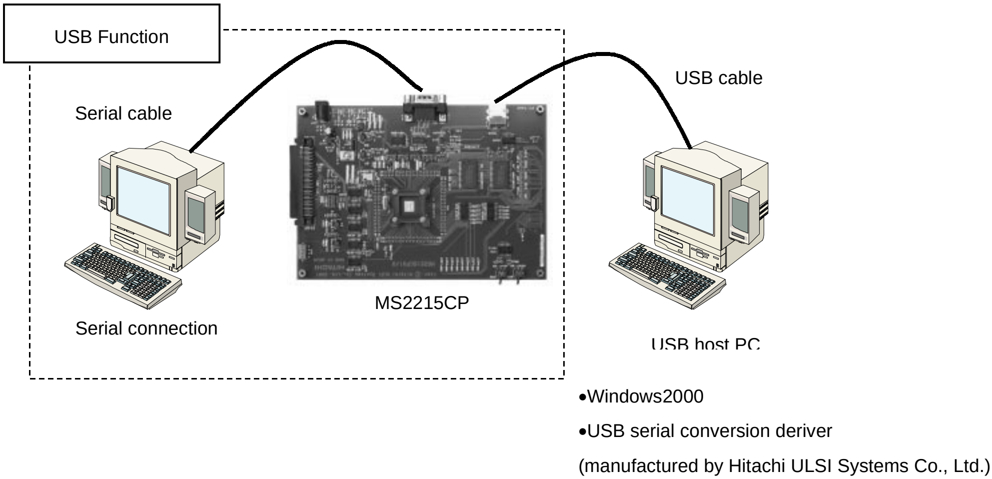


Figure 1.1 System Configuration Example

This system is configured of the H8S/2215 CPU board manufactured by Hitachi ULSI Systems Co., Ltd. (hereafter referred to as the MS2215CP) on which the H8S/2215 is mounted, a serially-connected PC, and a USB host PC (Windows 2000) containing the USB serial conversion driver*[1] (manufactured by Hitachi ULSI Systems Co., Ltd.).

In this system, the MS2215CP can receive the USB packet data transmitted from the USB host PC and transmit it to the serially-connected PC after converting it into serial data. Also, its reverse is possible, that is, the MS2215CP can receive serial data from the serially-connected PC and transmit it to the USB host PC after converting it into USB packet data.

This system offers the following features.

1. The sample program can be used to evaluate the USB module of the H8S/2215 quickly.
2. The sample program supports USB control transfer and bulk transport.
3. An E6000 (full-spec emulator) can be used, enabling efficient debugging.
4. Additional programs can be created to support interrupt transfer and isochronous transfer.*[2]

Notes: 1. For inquiries on this system (sample program and USB serial conversion driver), contact your Hitachi sales agency.
The USB serial conversion driver operates only with a vendor ID of 045B manufactured by Hitachi, Ltd. To use the USB serial conversion driver in your product, a contract concerning the USB serial conversion driver must be separately made with Hitachi ULSI Systems Co., Ltd.

2. Interrupt transfer and isochronous transfer programs are not provided, and will need to be created by the user.

1.2 Purpose of this System

The price reduction of PCs has been accelerated in recent days, and at the same time, the legacy-free PCs (equipped only with new standard ports compliant to Plug & Play such as USB (Universal Serial Bus), but not with old standard ports such as a serial port) have started to arrive on the market in large numbers. With this market trend, it may become impossible for the existing serial devices to be connected with PCs and many existing serial devices to be used. In order to solve this problem, a device which converts the existing serial line into the USB is required.

These application notes aim at providing an example of realizing the USB serial conversion function to solve this problem.

In this system, the USB does not exist when seen from the existing serial application. This is realized by providing the serial API when the existing serial devices are replaced by the new USB devices. This allows the application program to be used without changes.

Figure 1.2 shows the hardware and software configurations when the PC and serial devices are connected via the existing serial line. Figure 1.3 shows the hardware and software configurations when the PC and serial devices are connected via the USB serial conversion device.

As shown in figure 1.2 (a), the serial devices are connected to the PC via the serial cable in the existing system. However, as shown in figure 1.3 (a), the USB serial conversion device is required between the PC and serial devices when the existing serial devices are connected to the PC via the USB. The USB serial conversion device has a function to convert USB signals and serial signals mutually. The PC and USB serial conversion device are connected by the USB cable, and the USB serial conversion device and serial devices are connected via the serial cable. This makes it possible for the PC and serial applications to communicate with each other.

Figure 1.2 (b) and 1.3 (b) show the software configuration expressed in hierarchical structure. The connection indicated by a dotted line shows the image of logical connection.

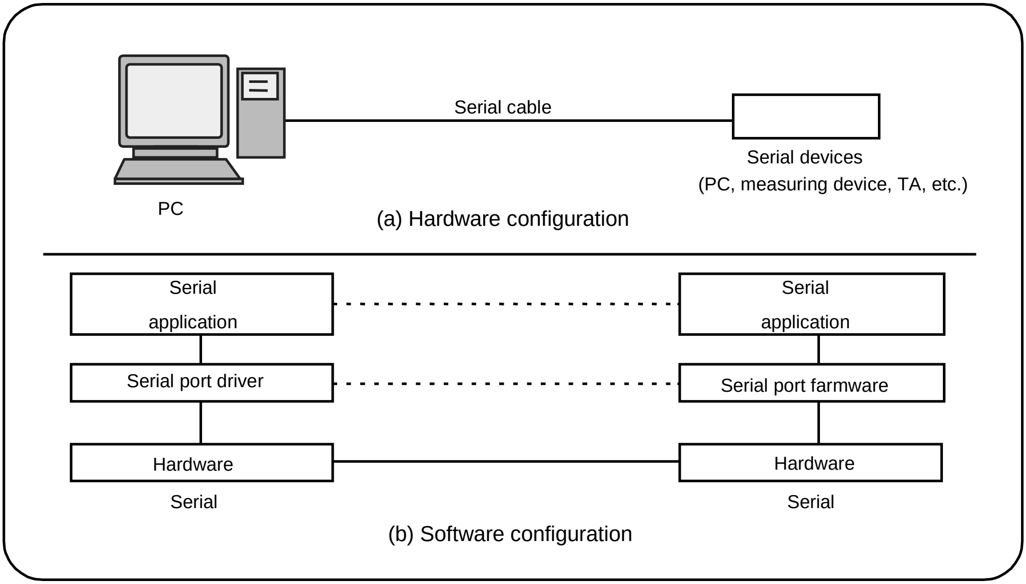


Figure 1.2 Example of Connecting PC and Serial Devices via Existing Serial Line

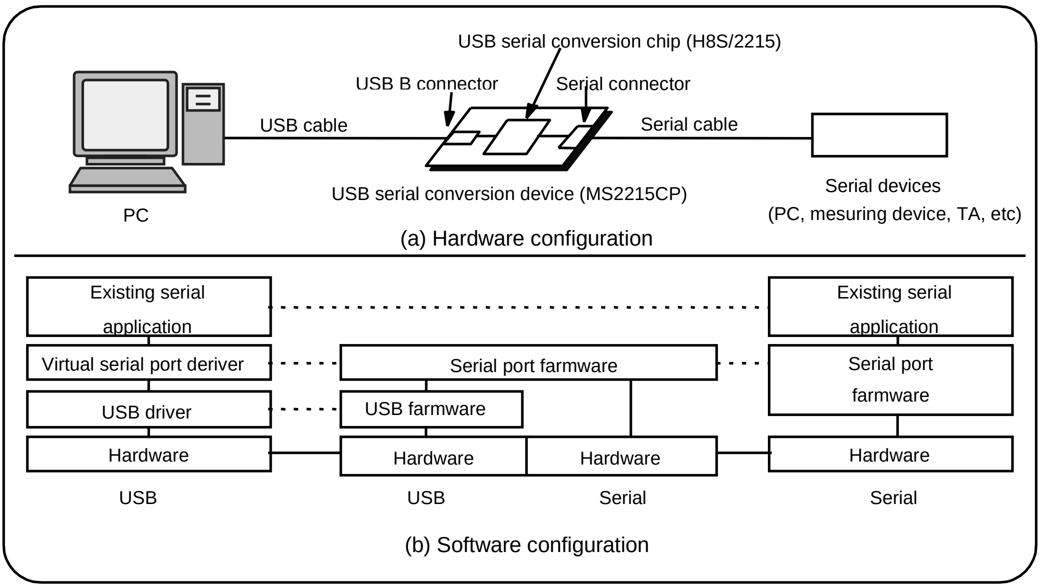


Figure 1.3 Example of Connecting PC and Serial Devices via USB

In figure 1.2 (b), transmit data from the serial application in the PC is sent to the serial port driver, which then sends the data to the serial hardware of the PC. The serial hardware sends this data to the serial hardware of the other end via a serial line. The serial port firmware of the serial device extracts the data from the hardware that received the data and sends it to the serial application. Herewith the data can be exchanged between serial applications.

As in figure 1.3 (b), the transmit data from the serial application in the PC is sent to the virtual serial port driver. This virtual serial port driver has the same application interface as the existing serial port driver. This allows the USB to not be recognized from the existing serial application, thus enabling data communication without having to change the existing serial application. The virtual serial port driver passes the data from the application to the lower USB driver. The USB driver then passes the data to the USB hardware in the PC. The USB hardware transmits the data through the USB bus to the USB hardware in the USB serial conversion device. The USB serial conversion device converts the received USB data into serial data and transmits it to the serial devices. The communication between the USB serial conversion device and serial devices has the same configuration as in figure 1.2. This makes it possible for the existing serial applications to exchange data with each other.

These application notes give an example for realizing the firmware operating on the MS2215CP, which is equivalent to the firmware in the USB serial conversion device in figure 1.3 (b).

Section 2 Development Environment

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

- H8S/2215 CPU board (type number: MS2215CP01-C/S) manufactured by Hitachi ULSI Systems Co., Ltd.
- H8S/2215 Series TFP-120 user system interface cable (hereafter called H8S/2215 user cable; type number: HS2215ECN61H) manufactured by Hitachi, Ltd.
- E6000 Emulator (type number: HS2214EPI61H) manufactured by Hitachi, Ltd.
- PC (Windows 95/98) equipped with an ISA (or PCI/PCMCIA) slot
- PC (Windows 2000) to serve as the USB host
- USB serial conversion driver manufactured by Hitachi ULSI Systems Co., Ltd.
- Serially-connected PC
- USB cable
- Serial cable (cross cable)
- Hitachi Debugging Interface (hereafter called HDI) manufactured by Hitachi, Ltd.
- Hitachi Embedded Workshop (hereafter called HEW) manufactured by Hitachi, Ltd.

2.1 Hardware Environment

Figure 2.1 shows device connections.

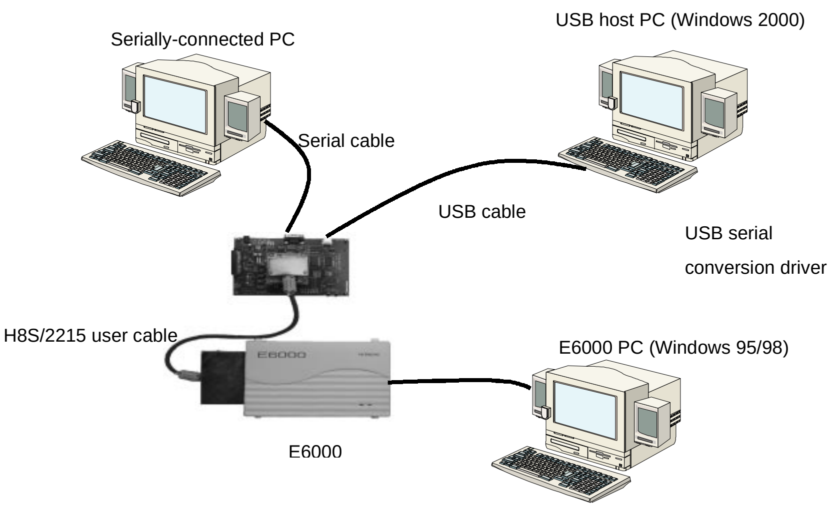


Figure 2.1 Device Connections

1. MS2215CP

   The jumper setting on the MS2215CP board shown in table 2.1 must be changed from that at shipment. Before turning on the power, ensure that the jumper is set as shown in table 2.1. There is no need to change any other jumpers or DIP switches.

Table 2.1 Jumper Setting

At Time of Shipment	After Change	Jumper Function
J9 1-2 closed	J9 2-3 closed	Switches PLLVCC pin level

2. USB host PC

   A PC with Windows 2000 installed, and with a USB port, is used as the USB host. A USB serial conversion driver (manufactured by Hitachi ULSI Systems Co., Ltd.) should be installed in this PC.

3. Serially-connected PC

   A PC with a serial port is used for transferring serial data.

4. E6000 PC

   The E6000 I/F board should be inserted into an ISA slot and connected to the E6000 via an interface cable. Then, the E6000 should be connected to the MS2215CP via an H8S/2215 user cable. After connection, start the HDI and perform emulation.

2.2 Software Environment

A sample program, the compiler and linker used, and the USB serial conversion driver are explained.

2.2.1 Sample Program

Files required for the sample program are all stored in the H8S2215 folder. When this entire folder with its contents is moved to a PC on which HEW and HDI have been installed, the sample program can be used immediately. Files included in the folder are shown in figure 2.2.

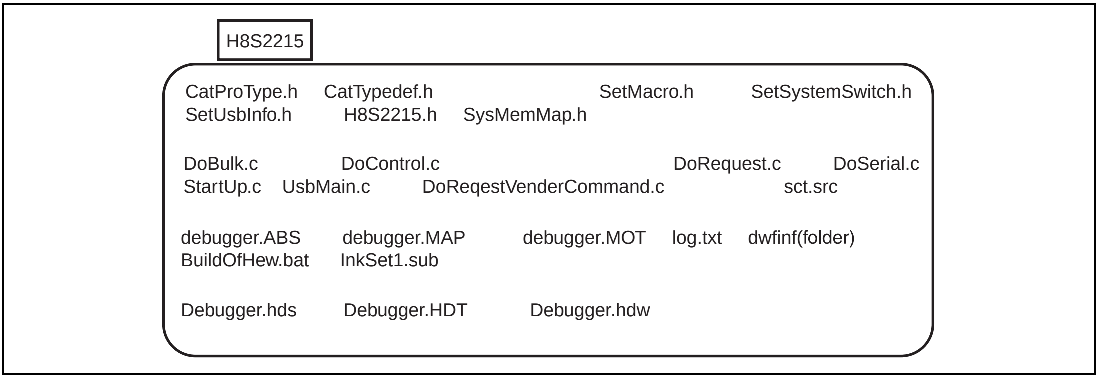


Figure 2.2 Files Included in H8S2215 Folder

2.2.2 Compiling and Linking

The sample program is compiled and linked using the following software.

Hitachi Embedded Workshop Version 1.0 (release 9) (hereafter called HEW)

When HEW is installed in C:\Hew*, the procedure for compiling and linking the program is as follows.

First, a folder named Tmp should be created below the C:\Hew folder for use in compiling (figure 2.3).

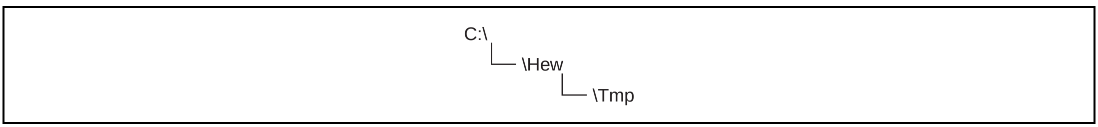


Figure 2.3 Creating a Working Folder

Next, the folder in which the sample program is stored (H8S2215) should be copied to C:\Usr (or can be copied to any location, then "C:\Usr\h8s2215" written in the debugger.hds file in the folder should be changed to the path to the copied folder). In addition to the sample program, this folder contains a batch file named BuildOfHew.bat. This batch file sets the path, specifies compile options, specifies a log file indicating the compile and linking results, and performs other operations. When BuildOfHew.bat is executed, compiling and linking are performed. As a result, a Motorola S-type format file named debugger.MOT, which is an executable file, is created within the folder. At the same time, a map file named debugger.MAP and a log file named log.txt are created. The map file indicates the program size and variable addresses. The compile results (whether there are any errors, etc.) are recorded in the log file.

Note: * If HEW is installed to a folder other than C:\Hew, the compiler path setting and settings for environment variables used by the compiler in BuildOfHew.bat, as well as the library settings in InkSet1.sub, must be changed. Here the compiler path setting should be changed to the path of ch38.exe, and the setting for the environment variable ch38 used by the compiler should be set to the folder of machine.h and the setting of ch38_tmp should specify the working folder for the compiler. The library setting should specify the path of c8s26a.lib.

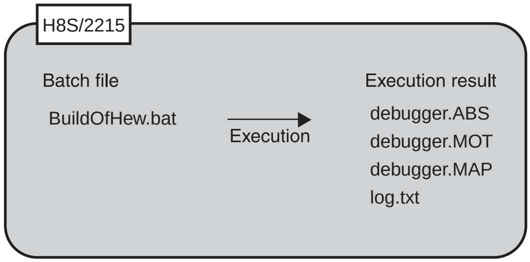


Figure 2.4 Compile Results

2.2.3 USB Serial Conversion Driver

Files required for the USB serial conversion driver are all stored in the UST-03 folder.

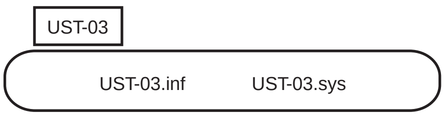
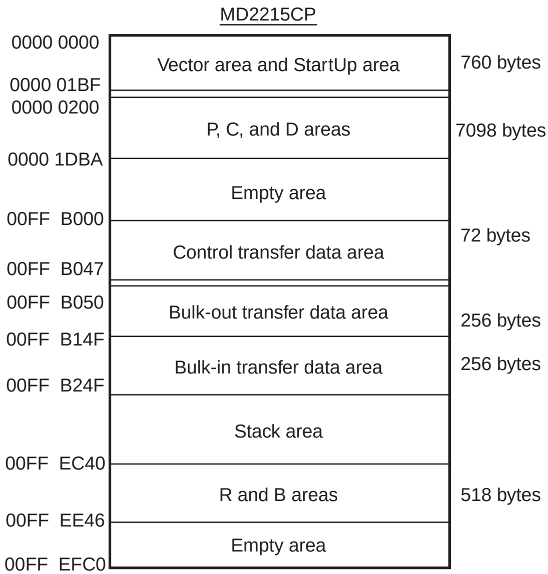


Figure 2.5 Files Included in UST-03 Folder

2.3 Loading and Executing the Program

Figure 2.6 shows the memory map for the sample program.



Note: The memory map differs according to the compiler version, compiling conditions, firmware upgrade, etc.

Figure 2.6 Memory Map

As shown in figure 2.6, this sample program allocates areas P, C, and D to on-chip flash memory, and areas R and B to the on-chip RAM area. These memory allocations are specified by the InkSet1.sub file in the H8S2215 folder.

2.3.1 Loading and Executing the Program

In order to load the sample program, the following procedure is used.

- Connect the E6000 PC in which the HDI has been installed to the E6000.
- Connect the E6000 to the MS2215CP via an H8S/2215 user cable.
- Connect the serially-connected PC to the MS2215CP via a serial cable.
- Turn on the power to the E6000 PC, serially-connected PC, and USB host PC for start up.
- Turn on the power to the E6000 and MS2215CP.
- Execute debugger.hds in the H8S2215 folder.

Through the above procedure, the sample program can be loaded into the MS2215CP.

After making the above settings, select Go from the Run menu to execute the program.

2.4 Method of Communication between PCs

2.4.1 Setting Up the USB Host PC

- Following the procedures in section 2.3.1, execute the sample program. When the sample program is activated properly, the 8-bit LED on the MS2215CP displays 0xAA.
- Insert a series B connector of the USB cable to the MS2215CP, and connect a series A connector on the opposite side to the USB host PC.
- The dialog box is displayed on the screen as below, and click “Next”.

- Select “Search for a suitable driver for my devide (recommended)”, and then click “Next”.

- Select “Floppy disk drives”, and then click “Next”.

- Make sure “UST-03.inf” is to be installed, and then click “Next”.

- Click “Finish”.

The installment of the driver has thus been completed and the MS2215CP is recognized as the serial COM port by the USB host PC.

Next, a hyper terminal, a communication software which is a standard attachment of WindowsOS, is initiated.

- Press the Windows key and select “Start → Program → Accessory (or under Communicaton)” to activate the hyper terminal.

- Input the file name (It can be random. USB-Serial has been input in the following screen.) and click “OK”.

- Select “COM3” for connection and click “OK”.

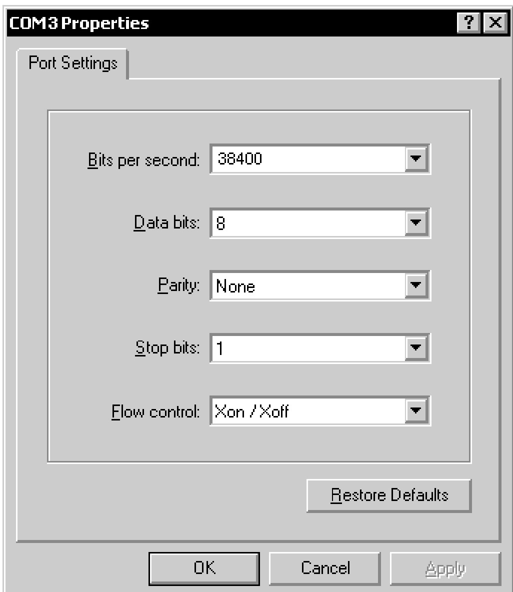
- The serial port is set within the range shown in table 2.2. The figure below is an example with the default values of this program entered. After the setting, click “OK”.

The hyper terminal has thus been initiated. If a value other than those shown in table 2.2 is entered, the 8-bit LED of the MS2215CP displays 0x30, and the default values of this program shown in table 2.2 are entered. If a value within the range is entered, the 8-bit LED keeps displaying 0xAA.

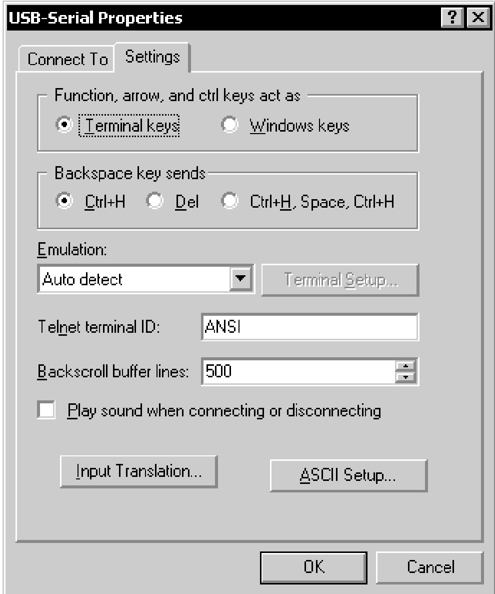
Table 2.2 Range of Possible Serial Port Settings

Item	Default Setting of This Program	Possible Settings
Bit/s [bps]	38400	9600, 19200, 38400*
Data bits	8	8 or 7
Parity	None	None, odd number, even number
Stop bit	1	1 or 2
Flow control	Xon/Xoff	Only Xon/Xoff

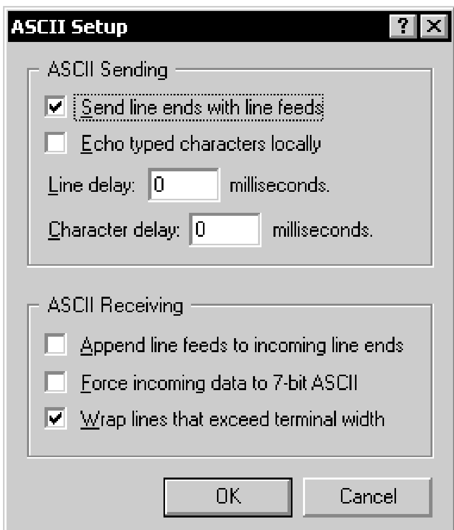
Note: * Since this sample program operates the CPU at 16 MHz, the error with a setting of 57600 bps or 115200 bps is too large, and may cause erroneous operation. Though a setting of 57600 bps or 115200 bps is possible in this sample program, the operation for such kind of a setting is not guaranteed.



- After the hyper terminal has been initiated, and before the communication begins, select “File Menu → Property → Setting” and click “ASCII Setup…”.



- Check the box for “Send line ends with line feeds” in ASCII Sending and then click “OK”.



2.4.2 Setting Up the Serially-Connected PC

The hyper terminal is initiated similarly as with the USB host PC. Make sure to enter the same values as the USB host PC to set the serial communication (bit/s, data bits, parity, stop bit, and flow control).

2.4.3 Communication between PCs

Once the hyper terminals for both the USB host PC and serially-connected PC are initiated, the characters input from the keyboard, text files, and binary files can be exchanged between the two PCs.

The characters input from the keyboard of the USB host PC side are transferred to the serially-connected PC. Also, the characters input from the keyboard of the serially-connected PC side are transferred to the USB host PC.

The text files can be transmitted to the other by selecting “Transfer → Transfer of text file”.

After selecting “Transfer → Reception of file → ZMODEM” in the receiving PC to make the receiving PC wait for file reception, the text files and binary files can be transmitted to the receiving PC by selecting “Transfer → Transmission of file → ZMODEM” in the transmitting PC.

Note: These application notes use a hyper terminal as a serial application to run on the PC. When using other serial applications, whether operation is correct must be confirmed separately.
This sample program performs flow control (Xon/Xoff). Therefore, a protocol supporting flow control (Xon/Xoff), e.g. ZMODEM, must be selected for file transmission.

Section 3 Overview of Sample Program

In this section, features of the sample program and its structure are explained. This sample program runs on the MS2215CP, and initiates USB transfers by means of interrupts from the USB function module or branches from the main loop. In addition, it initiates serial transfer by means of interrupts from the SCI1 or branches from the main loop. Of the interrupts from the on-chip modules in the H8S/2215, there are three interrupts related to the USB function module: EXIRQ0, EXIRQ1, and IRQ6. However, this sample program uses only the EXIRQ0. Even though there are four interrupts related to the SCI1 module: ERI1 (reception error), RXI1 (receive data full), TXI1 (transmit data empty), and TEI1 (transmit end), this sample program uses two interrupts: ERI1 and RXI1.

Features of this sample 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.
- Serial data can be received from the serially-connected PC.
- Serial data can be sent to the serially-connected PC.
- Serial transfer can be used to send data received by bulk-out transfer.
- Bulk-in transfer can be used to send data received serially.

3.1 State Transition Diagram

Figure 3.1 shows a state transition diagram for this sample program. In this sample program, as shown in figure 3.1, there are transitions between four states.

- Reset State

  Upon power-on reset and manual reset, this state is entered. In this reset state, the H8S/2215 mainly performs initial settings.
- Stationary State

  When initial settings are completed, a stationary state is entered in the main loop. In this stationary state, the data from the USB host PC and the serially-connected PC are monitored all the time, and if a data is detected, it is output to each of the other end PC. In other words, input data to the MS2215CP is monitored constantly, and if a data is detected, it is output to each of the other end PC.

- 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 the type of interrupt. The interrupt sources used in this sample program are indicated by the interrupt flag registers 0 to 3 (UIFR0 to UIFR3), and there are five interrupt sources in all. When an interrupt source occurs, the corresponding bits in UIFR0 to UIFR3 are set to 1.

- Serial Communication State

  In the stationary state, when an interrupt from the SCI1 module occurs, this state is entered. The interrupt sources used in this sample program are indicated by the serial status register (SSR1), and there are two interrupt sources in all: ERI1 and RXI1.

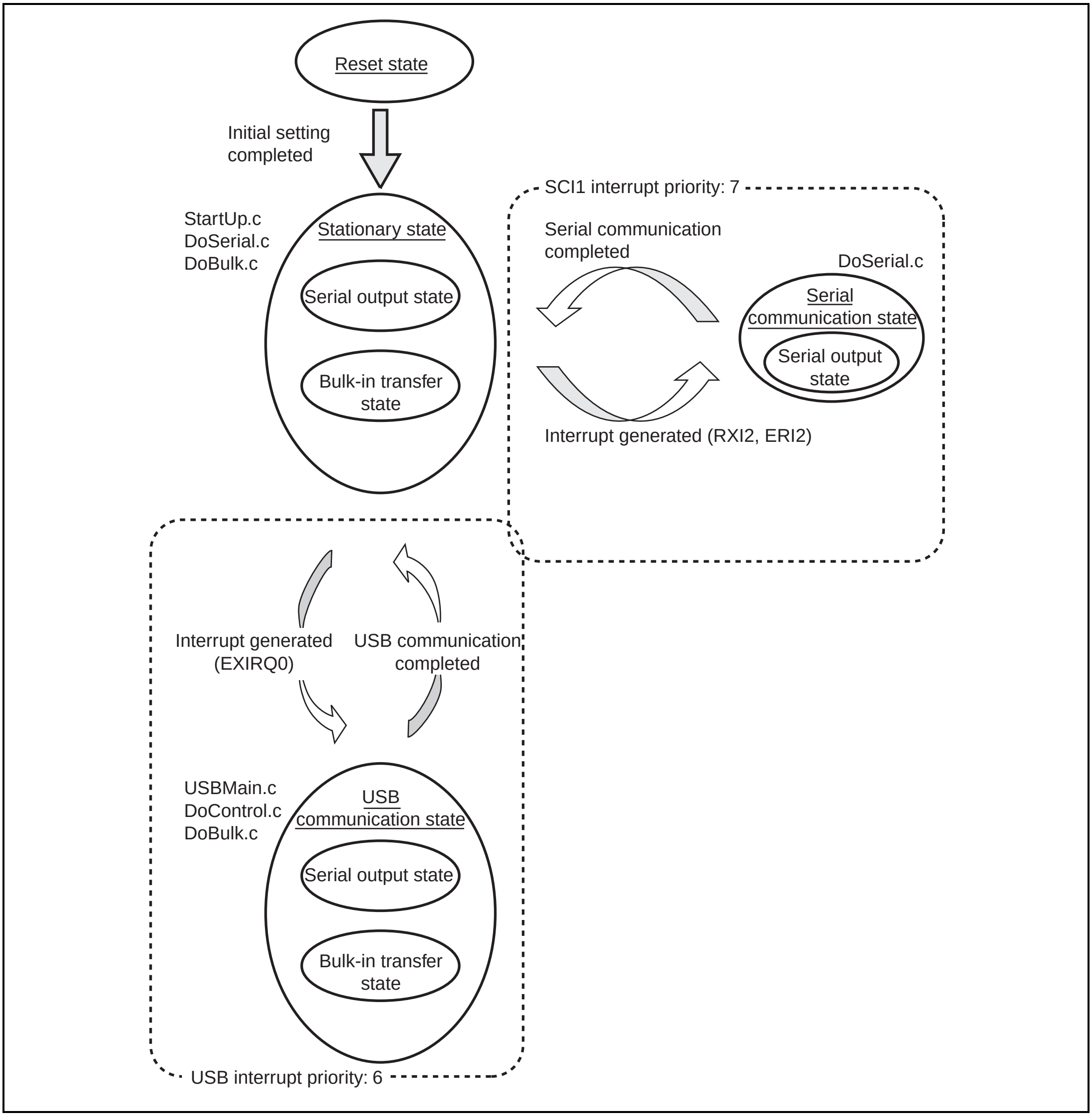


Figure 3.1   State Transition Diagram

In this sample program, the interrupt priority of the USB is set to 6 and that of the SCI1 to 7. This setting does not accept the USB interrupt during the SCI1 interrupt processing and prevents the serial receive processing from being delayed by the USB interrupt.

3.2 Overview of Communication between PCs

Figure 3.2 shows the overview of the communication between PCs. In this sample program, there are roughly two kinds of communication modes; USB communication and serial communication. Considering the data transmission and reception, the USB communication can be categorized by bulk-in and bulk-out transfer, and the serial communication can be categorized by serial input and serial output. Therefore there are a total of four communication modes in this sample program.

The data flow in this sample program can be categorized by two directions; from bulk-out transfer to serial output, and from serial input to bulk-in transfer, each of which is given 256-byte buffer. The input to the buffer of each direction handles interrupt operation and the output from the buffer controls the output on branching from the main loop. In the main loop, the RAM area for bulk-in/bulk-out transfers, which is a buffer for both directions, is monitored consistantly and, if any data exist, it is output from the buffer.

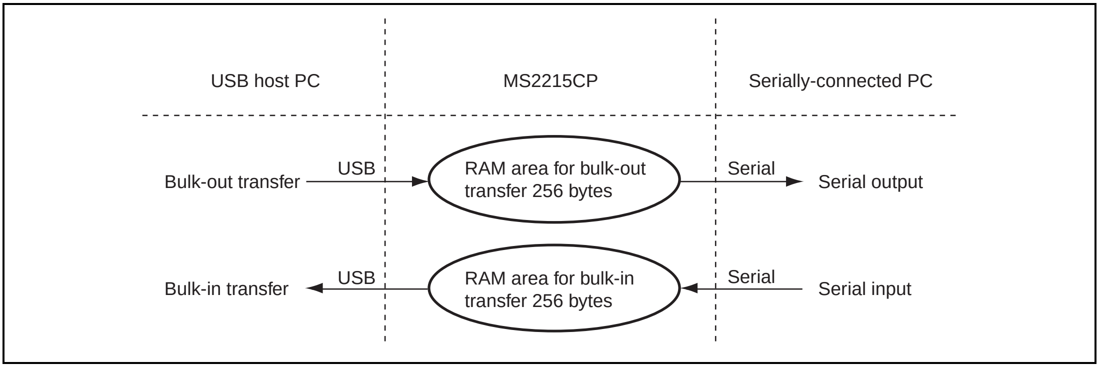


Figure 3.2 Communication between PCs

3.3 File Structure

This sample program consists of seven source files and seven header files. The overall file structure is shown in table 3.1. Each function is arranged in one file by transfer method or function type.

Table 3.1 File Structure

File Name	Principle Role
StartUp.c	Vector table settings, microcomputer initial setttings, and clearing ring buffer
DoSerial.c	Executing serial transmission/reception., and controlling SCI1 module
UsbMain.c	Determination of interrupt sources, and sending and receiving packets
DoRequest.c	Processing setup command issued by the host
DoControl.c	Executing control transfer
DoBulk.c	Executing bulk transfer
DoRequestVenderCommand.c	Processing vendor command
SysMemMap.h	Defining MS2215CP memory map addresses
SetUsbInfo.h	Defining USB structure
SetMacro.h	Defining macros
SetSystemSwitch.h	System operation settings
H8S2215.h	Defining H8S/2215 registers
CatTypedef.h	Defining structures
CatProType.h	Prototype declarations

3.4 Purposes of Functions

Table 3.2 shows functions contained in each file and their purposes.

Table 3.2-1 UsbMain.c

File in Which Stored	Function Name	Purpose
UsbMain.c	BranchOfInt	Determination of interrupt sources, and call function according to interrupt
	GetPacket	Write data transferred from the host controller to RAM
	PutPacket	Write data for transfer to the host controller to the USB module
	SetControlOutContents	Overwrite data sent from the host
	BE2ByteRead	Convert 2-byte data to big endian
	LE2ByteRead	Convert 2-byte data to little endian
	ActBusReset	Clear buffer, flag, and FIFO on receiving bus reset
	SetUsbModule	Initial setting of USB module
	USBclear	Clear ring buffer and flag

In UsbMain.c, interrupt sources are determined 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 3.2-2 StartUp.c

File in Which Stored	Function Name	Purpose
StartUp.c	SetPowerOnSection	BSC settings, module and memory initialization, and shift to main loop
	_INITSCT	Copies variables that have initial settings to the RAM work area
	InitMemory	Clears RAM area used in bulk communicatuion
	InitSystem	Pull-up control of the USB bus
	Error	Shifts CPU to sleep mode when error occurs
	SciInit	SCI1 initialization
	Set_SMR	Initial setting of SMR1 of SCI1
	ActBusVcc	Processing when VBUS is received

When a power-on reset or manual reset is carried out, SetPowerOnSection of the StartUp.c file is called. At this point, the RAM area used for the H8S/2215 initial settings, control transfer, and bulk transfer is cleared.

Table 3.2-3 DoSerial.c

File in Which Stored	Function Name	Purpose
DoSerial.c	ActSerialOut	Data is read from the read pointer and passed to ExSerialOut by 1 byte as parameter
	ActSerialIn	Write serially-input data to the area for bulk-in transfers
	WriteBulkInArea	Write data to the area for bulk-in transfers
	ExSerialOut	1-byte data is serially output from SCI1

In DoSerial.c, serial transmission and reception are executed as well as SCI1 module control.

Table 3.2-4 DoRequest.c

File in Which Stored	Function Name	Purpose
DoRequest.c	DecStandardCommands	Decode command issued by host controller, and process those which are standard commands

During control transfer, commands sent from the host controller are decoded, and commands are processed. In this sample program, a vendor ID of 045B (vendor: Hitachi) is used. When the customer develops a product, the customer should obtain a vendor ID at the USB Implementers' Forum.

Table 3.2-5 DoControl.c

File in Which Stored	Function Name	Purpose
DoControl.c	ActControl	Carries out the setup stage of control transfer
	ActControlIn	Carries out the data stage and status stage of control IN transfer (transfer in which the data stage is in the IN direction)
	ActControlOut	Carries out the data stage and status stage of control OUT transfer (transfer in which the data stage is in the OUT direction)

When the control transfer interrupt (EP0oTS) is generated, ActControl obtains the command, and decoding is carried out by DecStandardCommands. Next, the data stage and status stage are carried out using either ActControlIn or ActControlOut, depending on the type of command.

Table 3.2-6 DoBulk.c

File in Which Stored	Function Name	Purpose
DoBulk.c	ActBulkOut	Controls bulk-out-transfer
	ActBulkIn	Controls bulk-in transfer
	ExBulkOut	Execute GetPacket
	ExBulkIn	Execute PutPacket

These functions carry out processing involving bulk transfer as well as sending and receiving the data, and controlling the flow.

Table 3.2-7 DoRequestVenderCommand.c

File in Which Stored	Function Name	Purpose
DoRequestVenderCommand.c	DecVenderCommands	Responds to vendor commands

These functions carry out processing according to the vendor commands. In this sample program, processing is executed for the four vendor commands supported by the USB serial conversion driver manufactured by Hitachi ULSI Systems Co., Ltd. For details, refer to section 4.8, Vendor Command.

Figure 3.3 shows the interrelations between the functions explained in table 3.2. The upper-side functions call the lower-side functions. Also, multiple functions may 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. In the SCI1 interrupt, ActSerialIn is called. Figure 3.3 shows the hierarchical relation of functions; there is no order for function calling. For information on the order in which functions are called, refer to the flowcharts in section 4, Sample Program Operation.

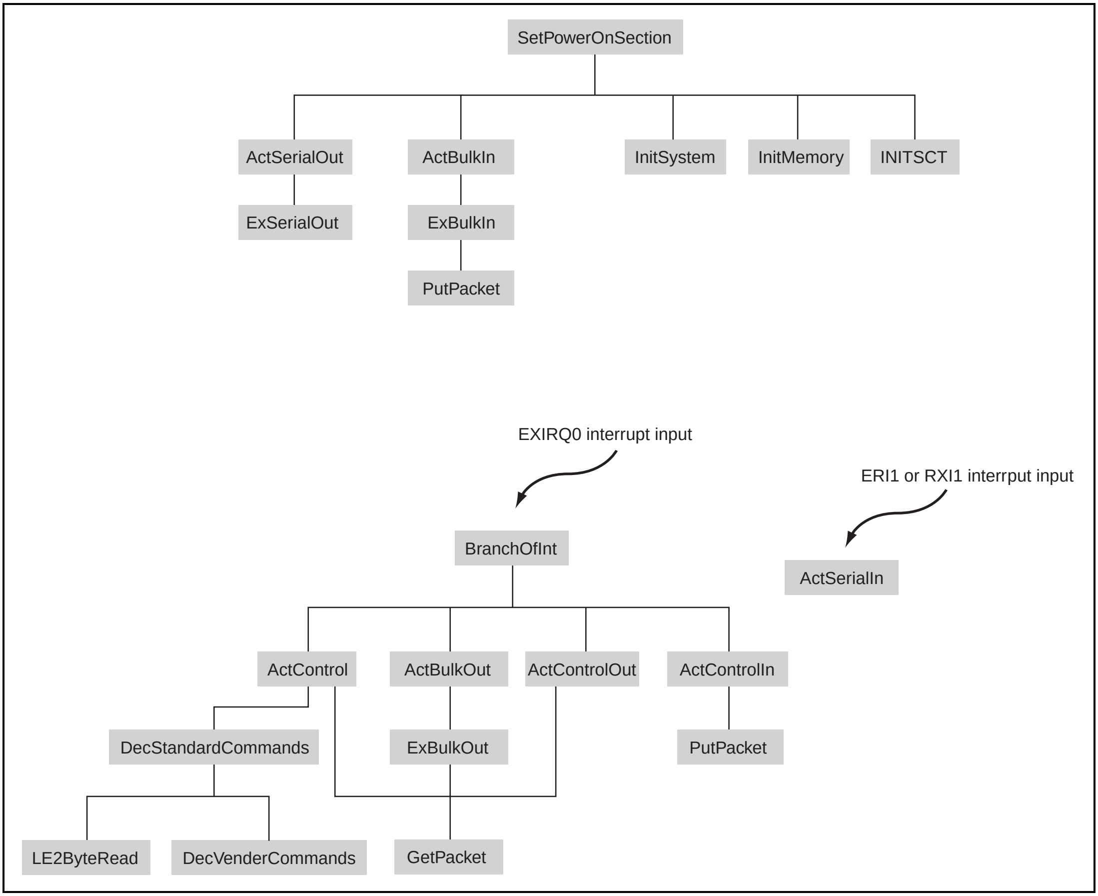


Figure 3.3 Interrelationship between Functions

Section 4 Sample Program Operation

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

4.1 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 4.1 is a flow chart for the SetPowerOnSection.

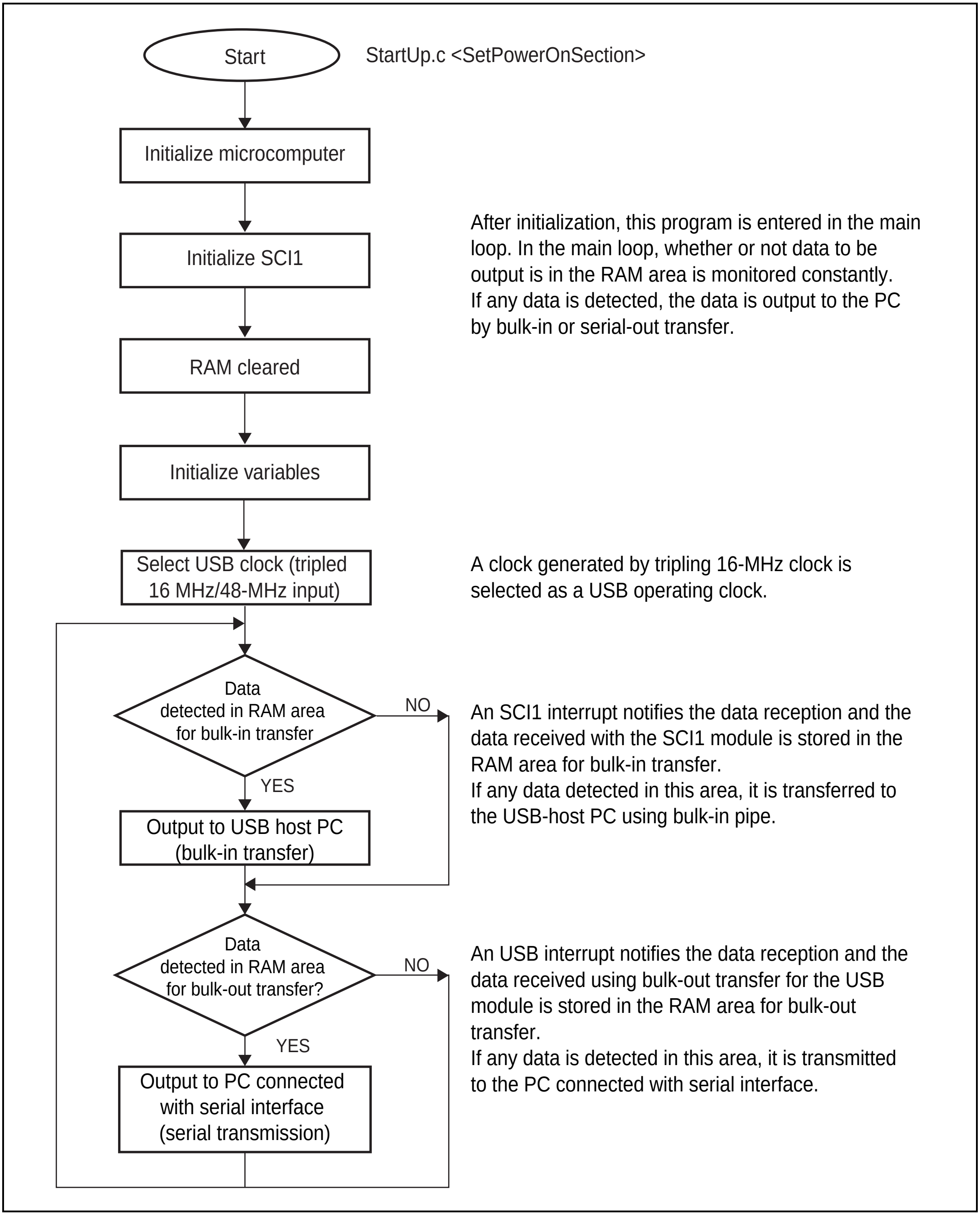


Figure 4.1 Main Loop

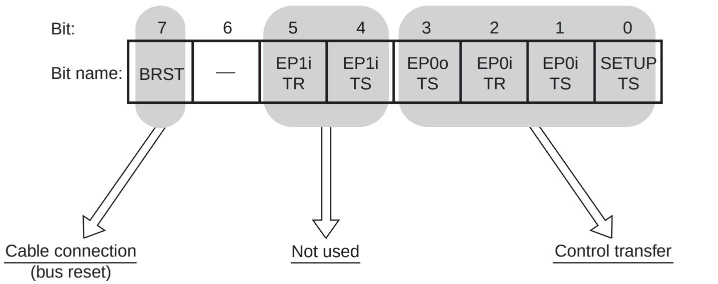
4.2 Types of Interrupts

As explained in section 3.1, the interrupts used in this sample program are indicated by the USB interrupt flag registers (UIFR0 to UIFR3) and serial status register (SSR1); there are five types of USB interrupts and two type of serial interrupts.

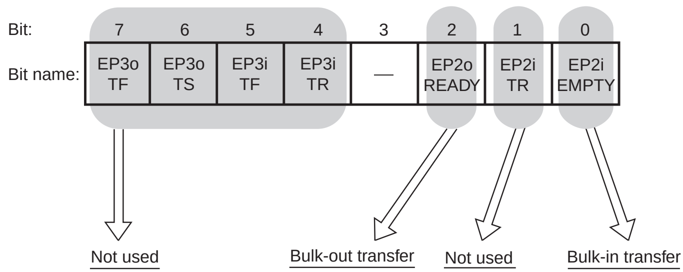
When a USB interrupt occurs, the corresponding bit in the interrupt flag register is set to 1 and a EXIRQ0 interrupt request is sent to the CPU. In the sample program, when the interrupt occurs, the CPU reads the interrupt flag register to perform the corresponding USB communication. Figure 4.2 shows correspondence between the interrupt flag registers and USB communications.

Bulk-in transfer is supported in this sample program. It, however, is enabled not by an interrupt operation, but by branching from the main routine. Therefore, bulk-in interrupt should be disabled and monitoring the EP2i EMPTY flag activates bulk-in transfer. The EP2i TR bit is not be used.

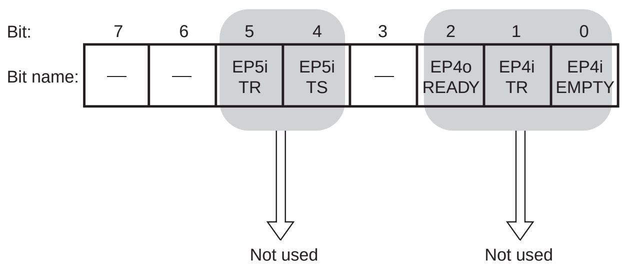
USB interrupt flag register 0 (UIFR0)



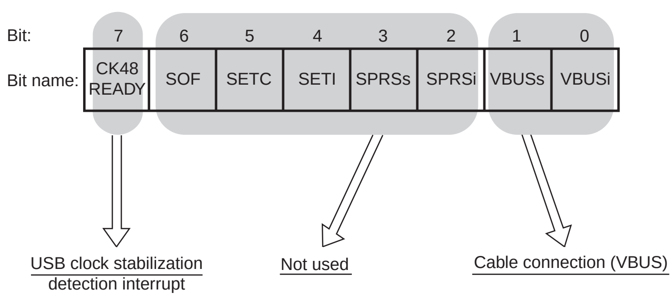
USB interrupt flag register 1 (UIFR1)



USB interrupt flag register 0 (UIFR2)



USB interrupt flag register 1 (UIFR3)



Note: This sample program does not support interrupt and isochronous transfers.

Figure 4.2   Types of USB Interrupt Flags

When an SCI1 interrupt occurs, the corresponding bit in the serial status register is set to 1 and an interrupt request is sent to the CPU. In this sample program, the transmit data empty and receive data full, that is, serial transmission and serial reception functions are supported. However, since the serial transmission is executed not by an interrupt operation, but by branching from the main loop, it is used only as a flag and the interrupt function is not used.

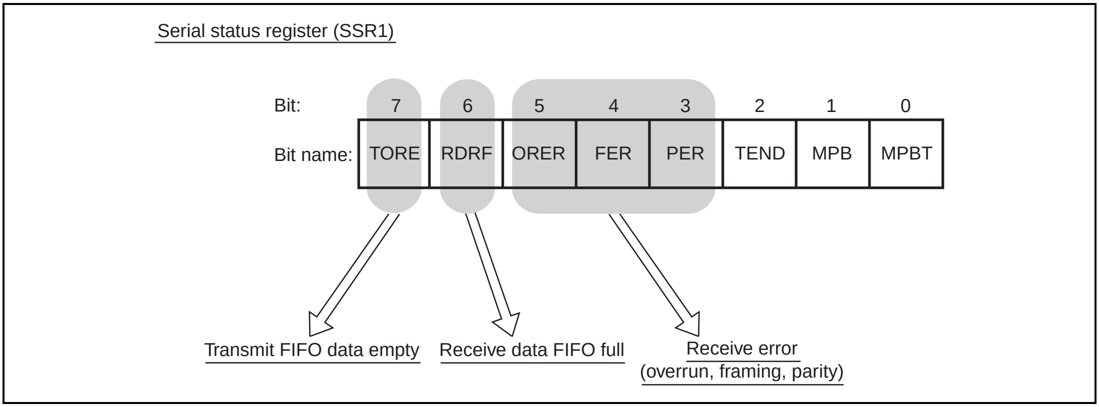


Figure 4.3   Types of Serial Interrupt Flags

4.2.1 Branching to Transfer Function

In this sample program, the transfer type is determined by method of calling each transfer function. The calling methods are a branch from the main loop and an interrupt from the USB function or SCI1 module. Table 4.1 shows correspondence between transfer types and methods of calling each transfer function.

When branching from the main loop, the function is directly called. This method corresponds to serial-out transfer (ActSerialOut) and bulk-in transfer (ActBulkIn). When branching by a USB interrupt, the branch is carried out by the BranchOfInt in UsbMain.c. This method corresponds to detection of USB operating clock stabilization (SetUsbModule), cable connection (ActBusReset, ActBusVcc), control transfer (ActControl) and bulk-out transfer (ActBulkOut). When branching by an SCI1 interrupt, the function is directly called because transfer functions are determined by interrupt sources in the SCI1 module, such as ERI2, RXI2 and TXI2. This method corresponds to serial-in transfer (ActSerialIn).

Table 4.1 Transfer Type and Method of Calling Function

Module	Transfer type	Method of calling
USB	Detection of USB operating clock stabilization time	USB interrupt
	Cable connection (bus reset)	USB interrupt
	Cable connection (BusVcc)	USB interrupt
	Control transfer	USB interrupt
	Bulk-out transfer	USB interrupt
	Bulk-in transfer	Branch from main loop
SCI1	Serial-in transfer	SCI1 interrupt
	Serial-out transfer	Branch from main loop

Table 4.2 shows the correspondence between the USB interrupt types and the function called by BranchOfInt.

Table 4.2 USB Interrupt Types and Called Functions

Register Name	Bit	Bit Name	Name of Function Called
UIFR0	0	BRST	ActBusReset
	1	—	—
	2	EP1i TR	—
	3	EP1i TS	—
	4	EP0o TS	ActControlIn, ActControlOut
	5	EP0i TR	ActControlOut
	6	EP0i TS	ActControlIn, ActControlOut
	7	SETUP TS	ActControl
UIFR1	7	EP3o TE	—
	6	EP3o TS	—
	5	EP3i TF	—
	4	EP3i TR	
	3	—	—
	2	EP2o Ready	ActBulkOut
	1	EP2i TB	—
	0	EP2i EMPTY	— (branch from main loop)
UIFR3	7	CK48 Ready	SetUSBModule
	6	SOF	—
	5	SETC	—
	4	SETI	—
	3	SPRSs	—
	2	SPRSi	—
	1	VBUSs	—
	0	VBUSi	ActBusVcc

The EP0i TS and EP0o Ts interrupts are used both for control-in and control-out transfers. 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 more details, refer to section 4.4, Control Transfers.

Table 4.3 shows SCI1 interrupt types and called functions.

Table 4.3 SCI1 Interrupt Types and Called Functions

Register Name	Bit	Bit Name	Name of Function Called
SSR1	7	TDRE	— (branch from main loop)
	6	RDRF	ActSerialOut
	5	ORER	ActSerialOut
	4	FER	ActSerialOut
	3	PER	ActSerialOut
	2	TEND	—
	1	MPB	—
	0	MPBT	—

From the next section, details of application-side firmware are explained for each USB and SCI1 transfer type.

4.3 Interrupt by Detection of USB Operating Clock Stabilization

This interrupt is generated when the 48-MHz USB operating clock stabilization time has been automatically counted after USB module stop mode cancellation. When endpoint information is written to UEPIR00_0 to UEPIR22_4 and each interrupt is specified after reception of the interrupt, the USB function module is entered in the USB cable-connection-wait state.

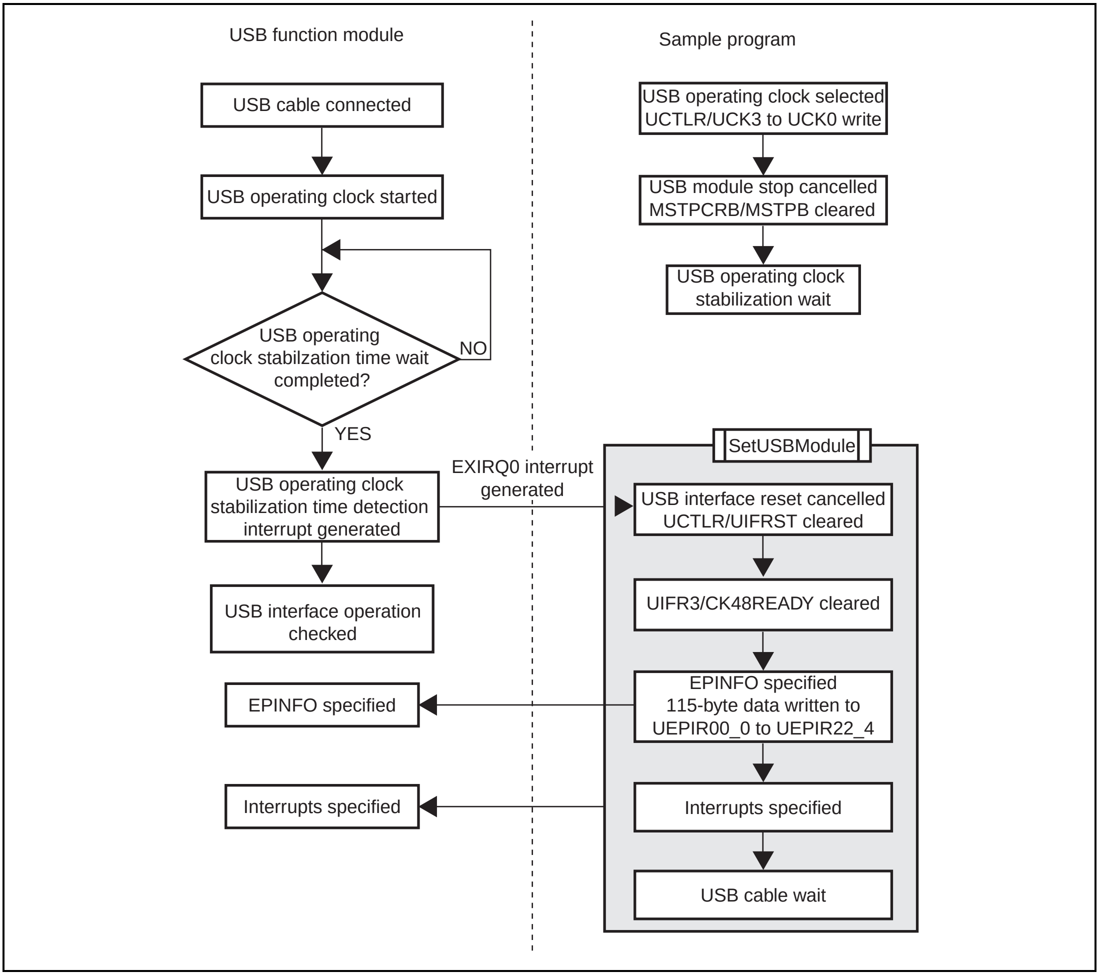


Figure 4.4 Interrupt at Detection of USB Operating Clock Stabilization

4.3.1 EPINFO

The USB function module incorporated in the H8S/2215 supports the following transfer types: one pipe for control transfer, two pipes for bulk-in transfer, two pipes for bulk-out transfer, two pipes for interrupt-in transfer, one pipe for isochronous-in transfer, and one pipe for isochronous-out transfer. The transfer types are shown in table 4.4. Any endpoint number, interface number, alternate number, and maximum packet size for each transfer type except for control transfer are specifiable.

Table 4.4 Correspondence between Transfer Type and UEPIR

Transfer Type	Number of Pipes	UEPIR
Control transfer	1	00
Interrupt-in transfer	2	01, 22
Bulk-in transfer	2	02, 20
Bulk-out transfer	2	03, 21
Isochronous-in transfer	1	04, 06, 08, 10, 12, 14, 16, 18
Isochronous-out transfer	1	05, 07, 09, 11, 13, 15, 17, 19

In this application note, endpoints are configured shown in figure 4.5.

The endpoint configuration in the H8S/2215 hardware manual is described in the way which conforms to Bluetooth standard. Their correspondence is shown in figure 4.5.

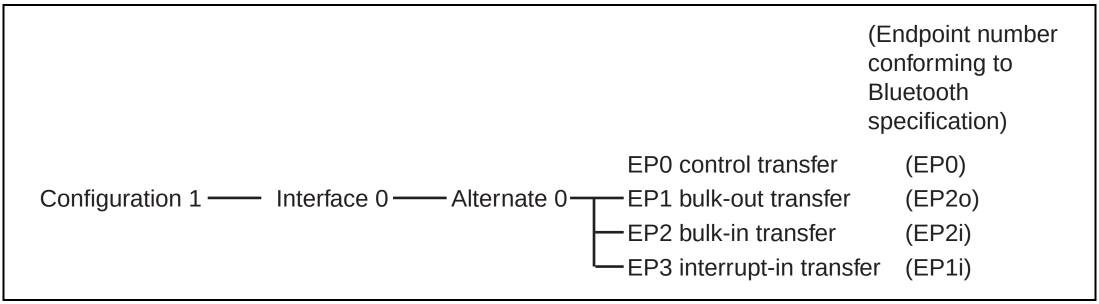


Figure 4.5 Endpoint Configuration for this Application Note

The settings for UEPIR00_0 to UEPIR22_4 are shown to configure the endpoints in figure 4.5. Dammy data should be written to endpoints which are not used.

RENESAS

Table 4.5 UEPIR Settings

UEPIR	Setting (Hex)	Transfer Type	EP Number	Interface Number	Alternate Number	MaxPacket Size (Byte)
00	00_00_40_00_00	Control	0	0	0	64
01	34_1C_08_00_01	Interrupt-in	3	0	0	8
02	24_14_40_00_02	Bulk-in	2	0	0	64
03	14_10_40_00_03	Bulk-out	1	0	0	64
04	04_1C_00_00_04	Isochronous-in	0	0	0	0
05	04_08_00_00_05	Isochronous-out	0	0	0	0
06	04_1C_00_00_06	Isochronous-in	0	0	0	0
07	04_08_00_00_07	Isochronous-out	0	0	0	0
08	04_1C_00_00_08	Isochronous-in	0	0	0	0
09	04_08_00_00_09	Isochronous-out	0	0	0	0
10	04_1C_00_00_0A	Isochronous-in	0	0	0	0
11	04_08_00_00_0B	Isochronous-out	0	0	0	0
12	04_1C_00_00_0C	Isochronous-in	0	0	0	0
13	04_08_00_00_0D	Isochronous-out	0	0	0	0
14	04_1C_00_00_0E	Isochronous-in	0	0	0	0
15	04_08_00_00_0F	Isochronous-out	0	0	0	0
16	04_1C_00_00_10	Isochronous-in	0	0	0	0
17	04_08_00_00_11	Isochronous-out	0	0	0	0
18	04_1C_00_00_12	Isochronous-in	0	0	0	0
19	04_08_00_00_13	Isochronous-out	0	0	0	0
20	04_14_00_00_14	Bulk-in	0	0	0	0
21	04_10_00_00_15	Bulk-out	0	0	0	0
22	04_10_00_00_16	Interrupt-in	0	0	0	0

4.4 Interrupt by Cable Connection (BRST, VBUS)

This interrupt occurs when a USB cable is connected to the host controller. After completion of initializing the microcomputer, the application side pulls up the USB data bus D+ using general-purpose output port. By means of this pull-up, the host controller detects that the device has been connected (figure 4.6).

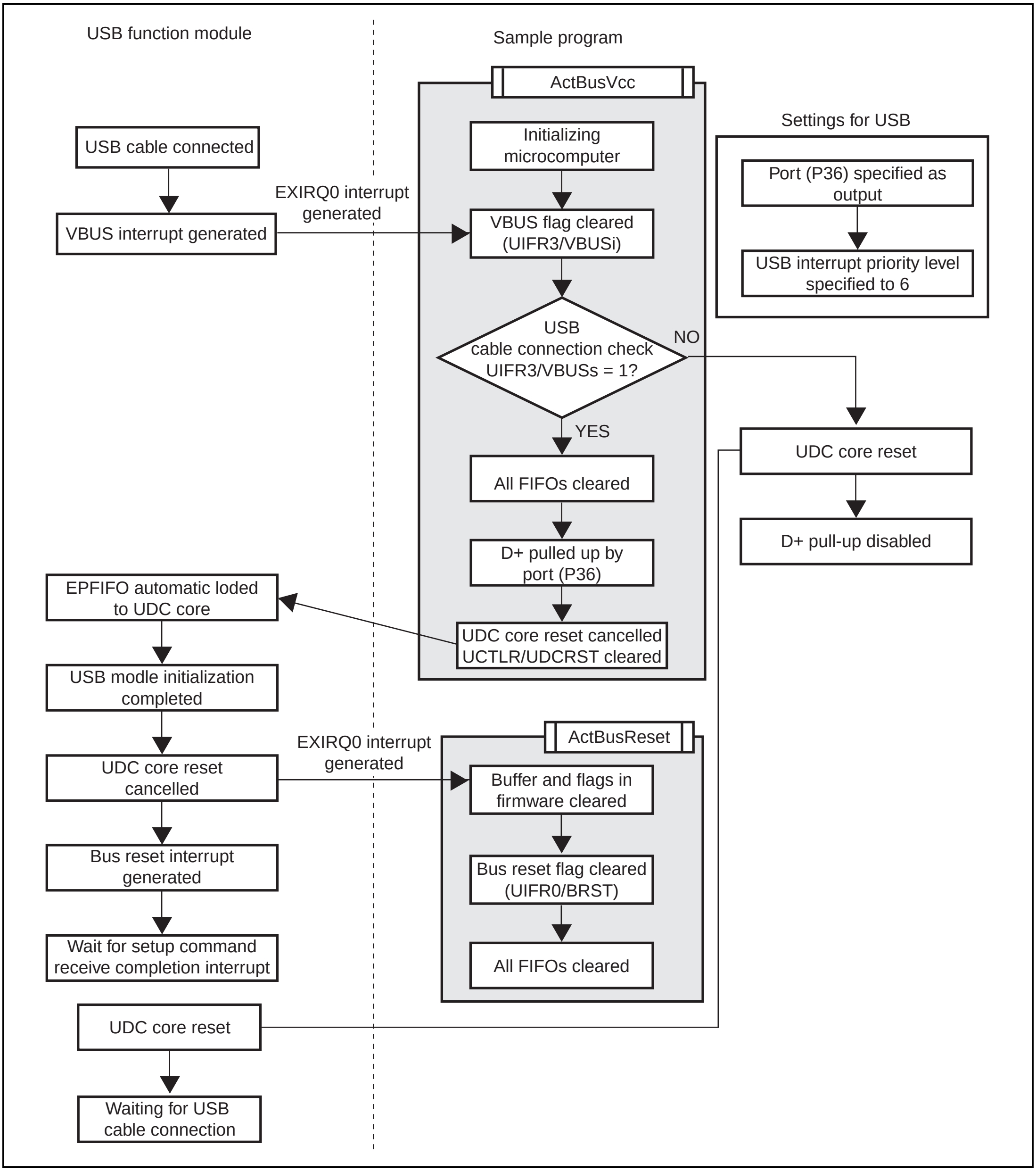


Figure 4.6 Interrupt by Cable Connection

4.5 Control Transfers

Control transfers are performed using bits 0 to 3 of the interrupt flag registers. Control transfers are divided into two types according to the direction of data in the data stage (see figure 4.7). In the data stage, data transfer from the host controller to the USB function module is control-out transfer and transfer in the opposite direction is control-in transfers.

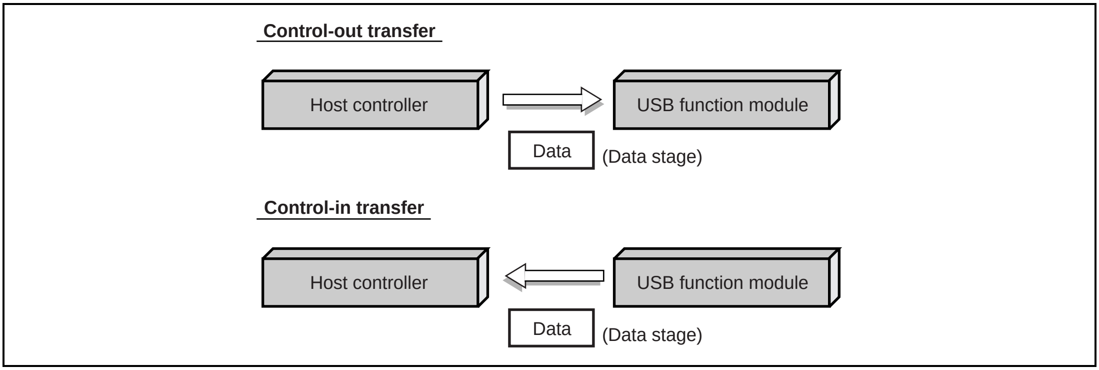


Figure 4.7 Control Transfers

Control transfers consist of three stages: setup, data (no data is possible), and status (see figure 4.8). Furthermore, a data stage consists of multiple bus transactions.

In control transfers, stage changes are detected by inverting the data direction. Hence the same interrupt flag for either control-in or control out transfer is used to call a function (see table 4.1). For this reason, the firmware must manage the control transfer type currently being performed, control-in or control-out transfer, in each state (see figure 4.8) and must call the appropriate function. States in the data stage (TRANS_IN, TRANS_OUT) are determined by commands received in the setup stage.

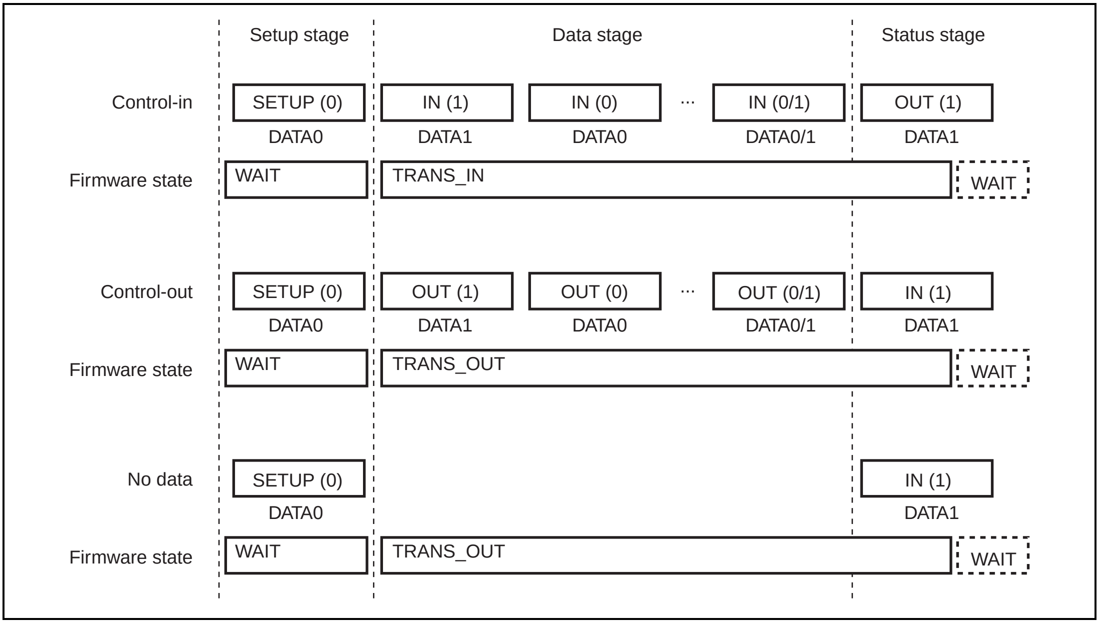


Figure 4.8 Stages in Control Transfers

4.5.1 Setup Stage

In the setup stage, commands are transferred between the host controller and USB function module. The firmware is entered in the WAIT state on both control-in and control-out transfers. Whether control-in transfer or control-out transfer is performed is determined by the type of the issued command and the state of the firmware in the data stage (TRANS_IN or TRANS_OUT) is also determined.

- Commands for control-in transfer: GetDescriptor (TRANS_IN) standard command
  GetLineCoding (TRANS_IN) vendor command
- Commands for control-in transfer: SetLineCoding (TRANS_OUT) vendor command
  SetControlLineState (TRANS_OUT) vendor command
  SendBreak (TRANS_OUT) vendor command

Figure 4.9 shows operation of the sample program in the setup stage. The figure on the left shows operation of the USB function module.

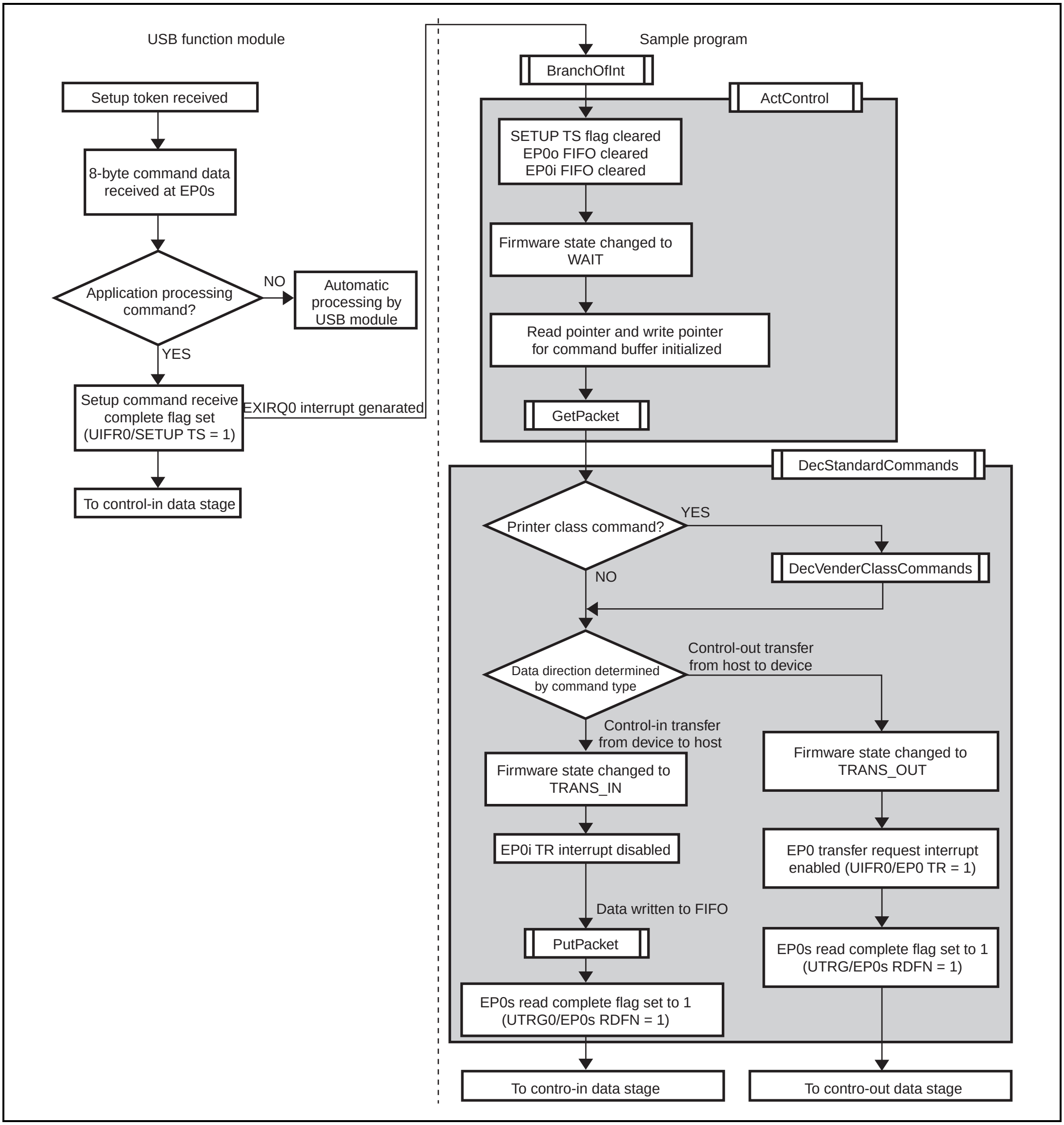


Figure 4.9 Setup Stage

4.5.2 Data Stage

In the data stage, data is transferred between the host controller and USB function module. The firmware is entered in the TRANS_IN state for control-in transfer or in the TRANS_OUT state for control-out transfer according to the result of decoding the command in the setup stage. Figures 4.10 and 4.11 show the operation of the sample program in the data stage on control transfers.

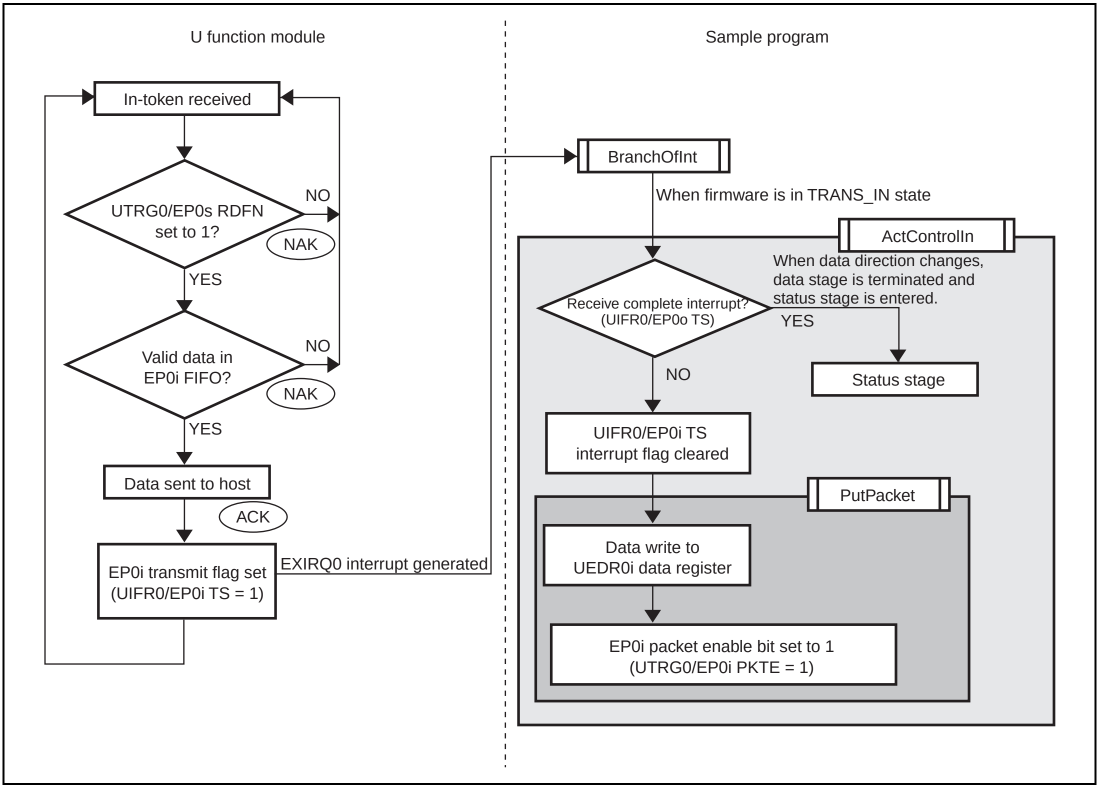


Figure 4.10 Data Stage (Control-In Transfer)

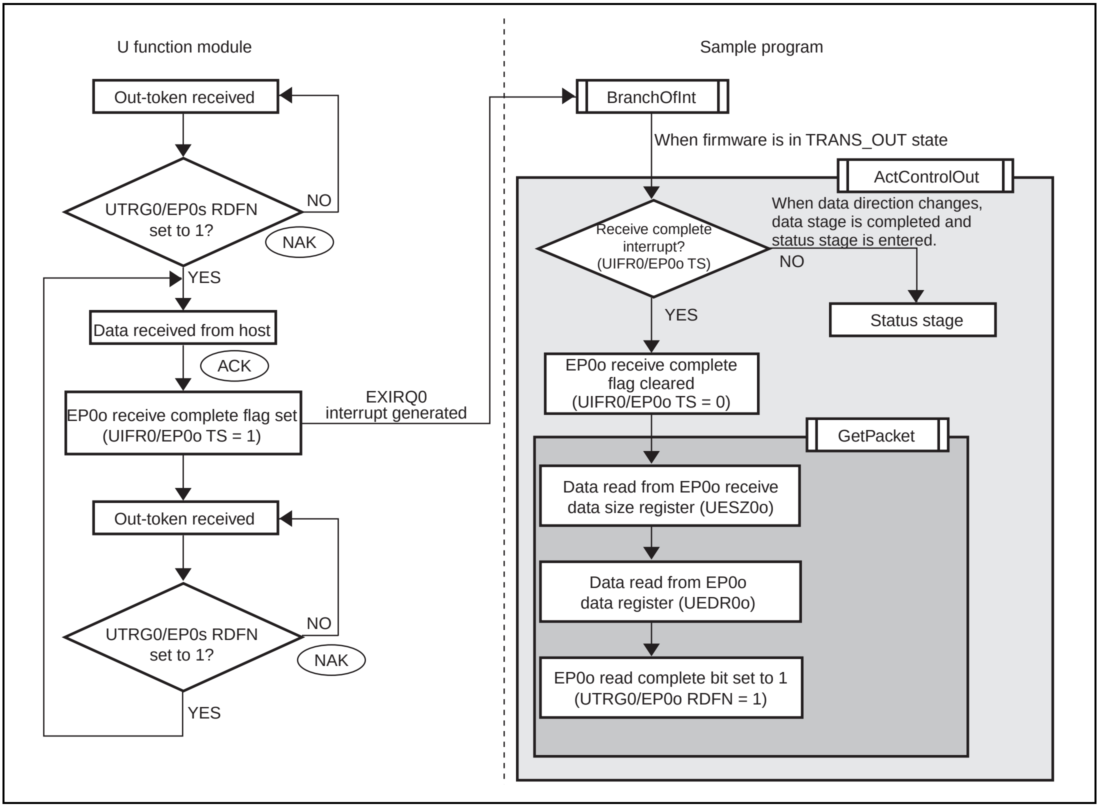


Figure 4.11 Data Stage (Control-Out Transfer)

4.5.3 Status Stage

The status stage is started by a token with the opposite direction of the data stage, that is, the status stage is started by an out-token from the host controller on control-in transfer and is started by an in-token from the host controller on control-out transfer.

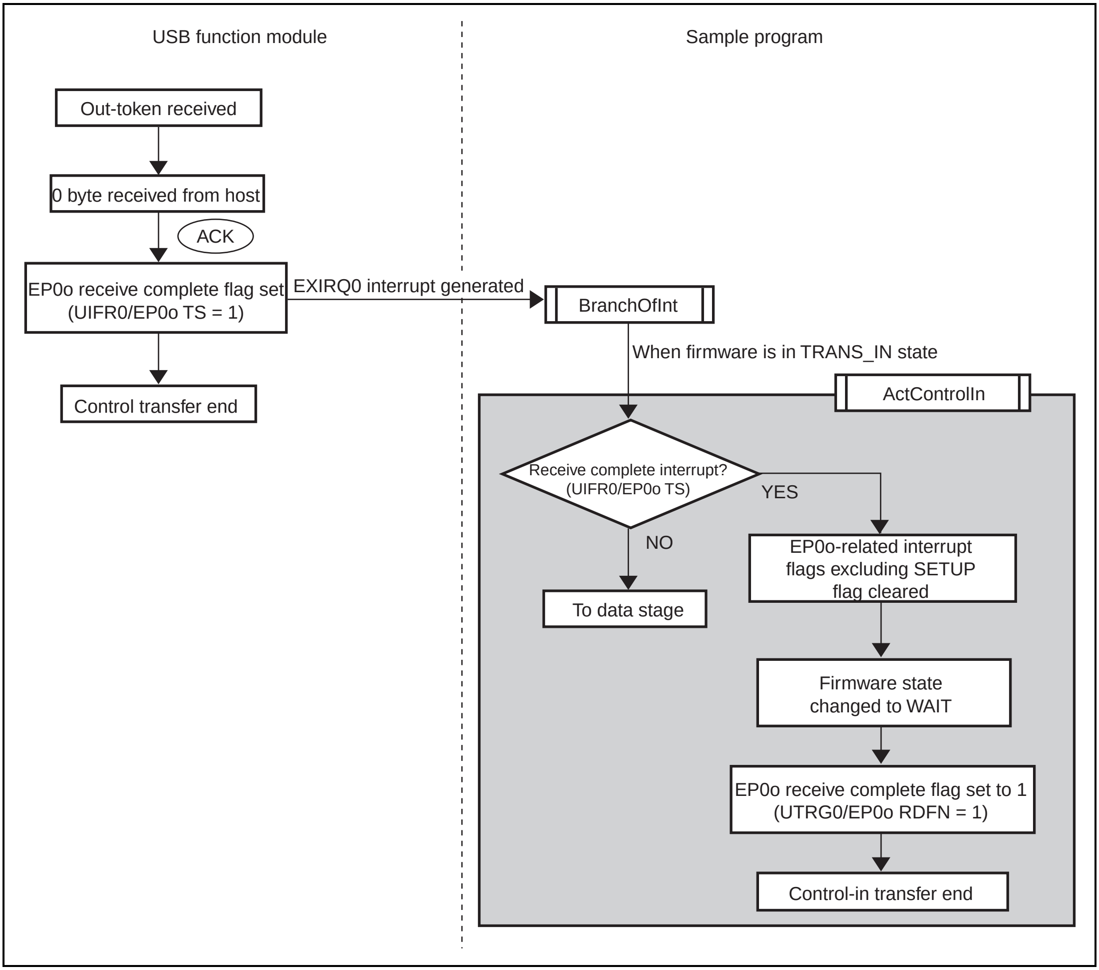


Figure 4.12 Status Stage (Control-In Transfer)

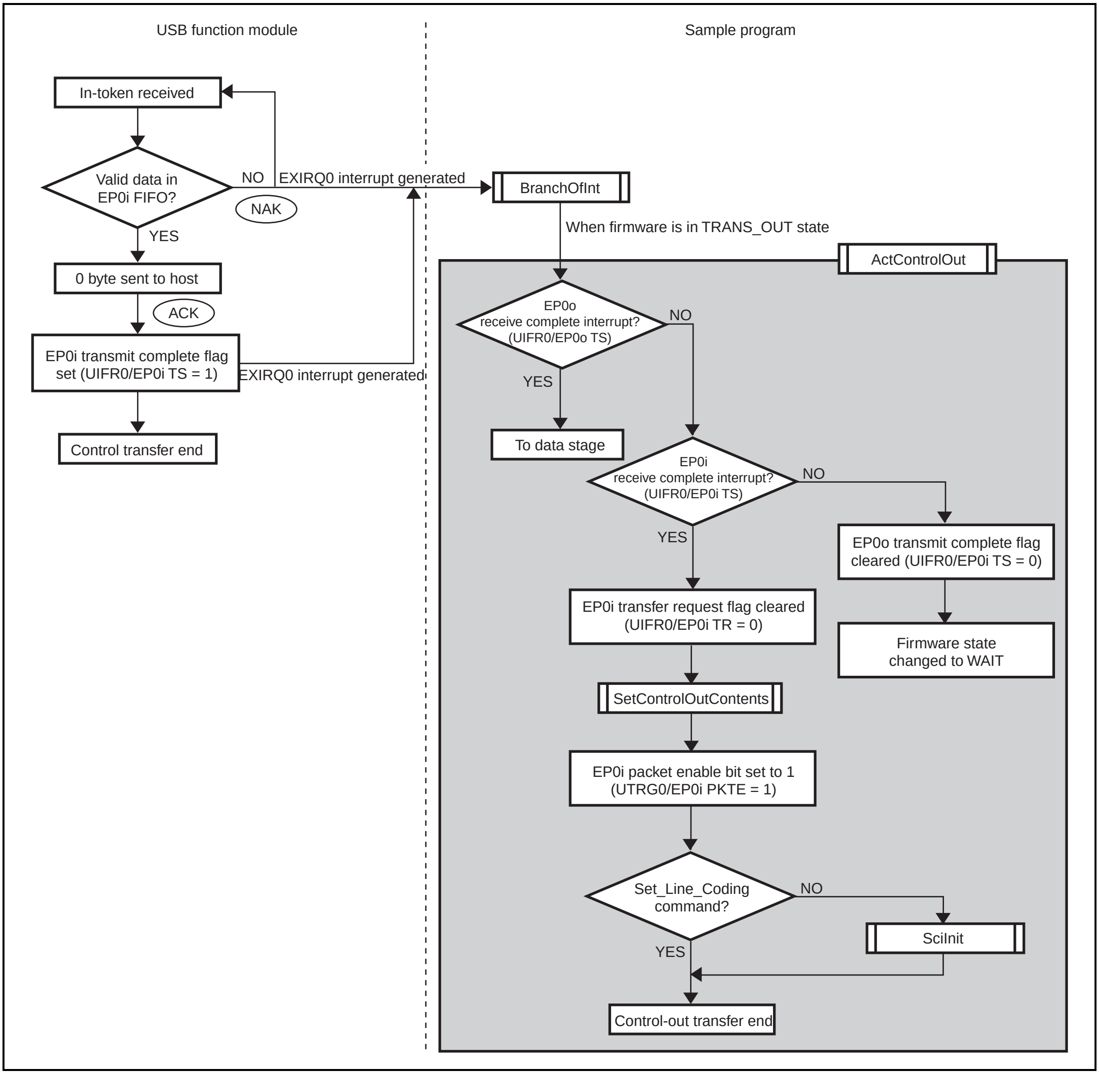


Figure 4.13 Status Stage (Control-Out Transfer)

4.6 Bulk Transfers

Bulk transfers are performed using bits 0 to 2 of the interrupt flag register 1 (bits 0 and 1 are not used because a bulk-in transfer is not enabled by an interrupt in this program). Bulk transfers are also be divided into two types according to the direction of data transfer (figure 4.14).

Data transfer from the host controller to the USB function module is bulk-out transfer and data transfer in the opposite direction is bulk-in transfer.

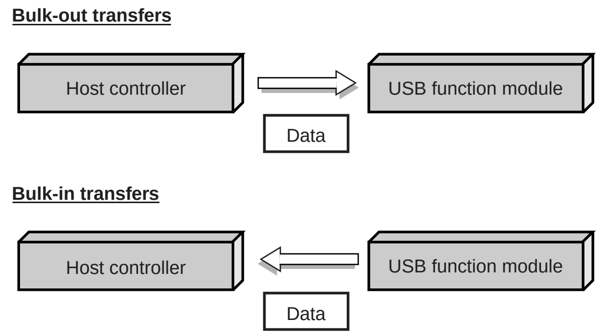


Figure 4.14 Bulk Transfers

4.6.1 Bulk-Out Transfers

Figure 4.15 shows the operations of the sample program when bulk-out transfer is carried out.

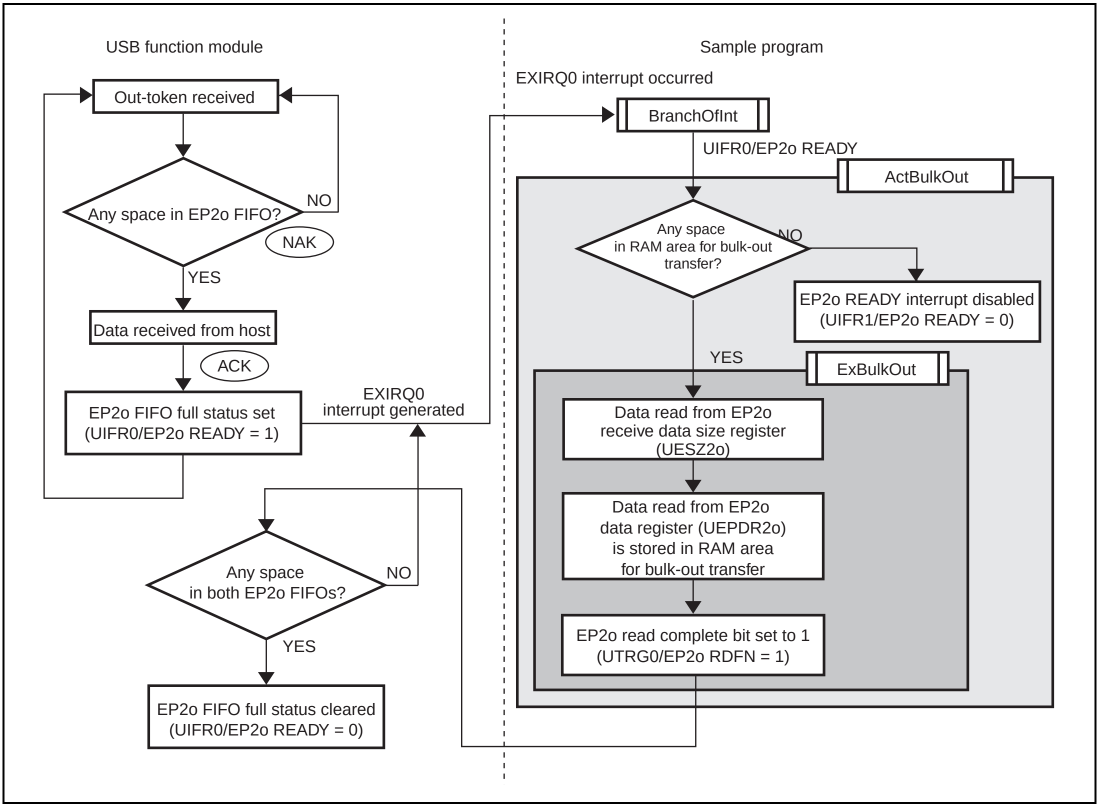


Figure 4.15 Bulk-Out Transfers

4.6.2 Bulk-in Transfers

Figure 4.16 shows the operation of the sample program when bulk-in transfer is carried out. Unlike bulk-out transfer, bulk-in transfer is not started by an interrupt and is started by a branch from the main loop.

When there is no space in the RAM area and the serial-in transfer is disabled, data stored in the RAM area for bulk-in transfer can be written to the UEDR2i data register. Whether or not the RAM area is made available by this write operation can be checked. When the RAM area is made available, serial-in transfer can be enabled.

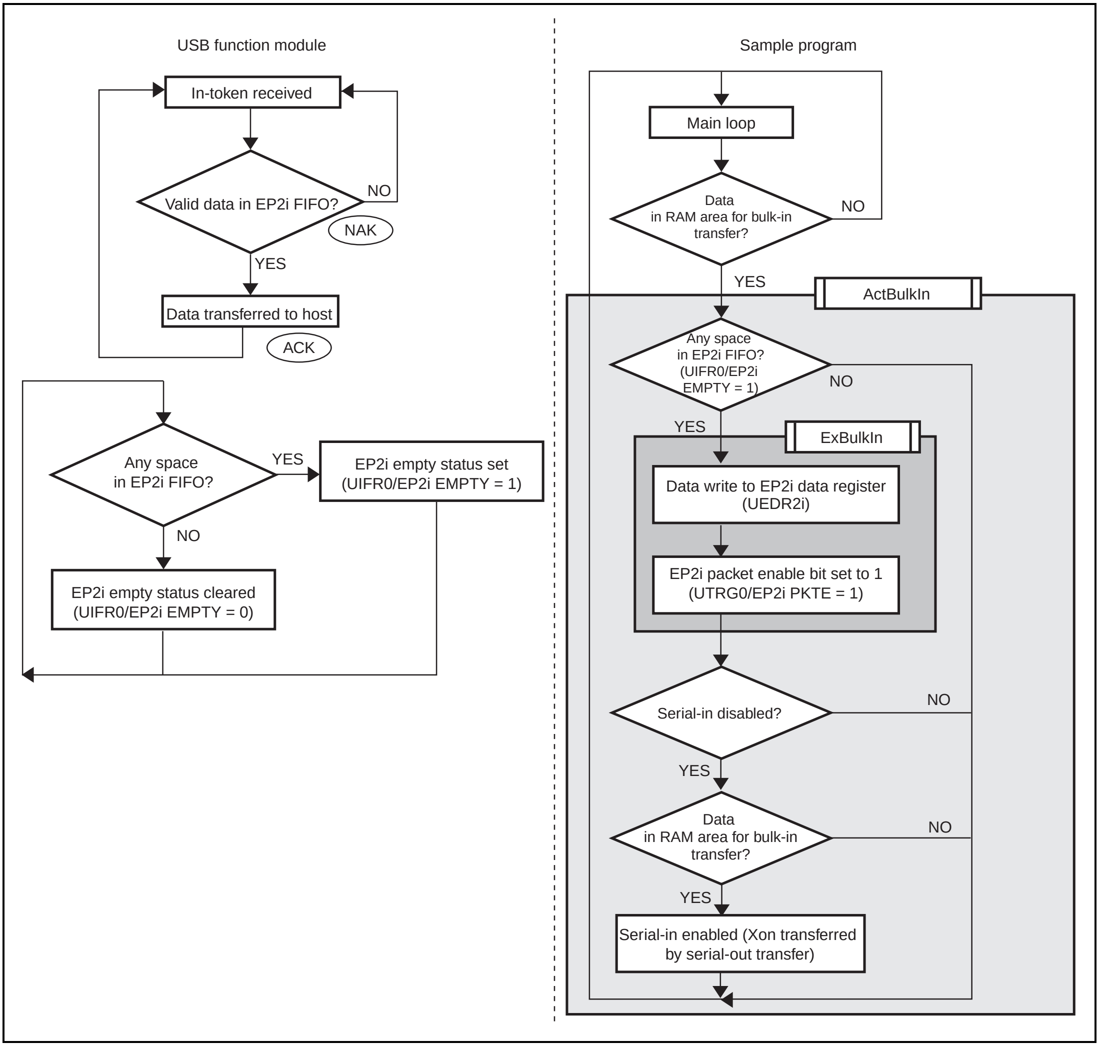


Figure 4.16 Bulk-In Transfer

4.7 Serial Transfer

The SCI1 module is used for serial transfer. Serial-out transfer is performed by branching from the main loop and serial-in transfer is performed by an interrupt. The RDRF flag of the serial status register (SSR1) is used on serial-in transfer.

4.7.1 Serial-Out Transfer

Figure 4.17 shows the operation of the sample program on serial-out transfer. When any data is in the RAM area for bulk-out transfer, the ActSerialOut function is called to branch from the main loop and the SCI1 module is used to transfer the data. When data is not in the RAM area for bulk-out transfer and the bulk-out transfer is disabled, whether or not the RAM area is made available by this serial-out transfer can be checked. When the RAM area is made available, bulk-out transfer can be enabled.

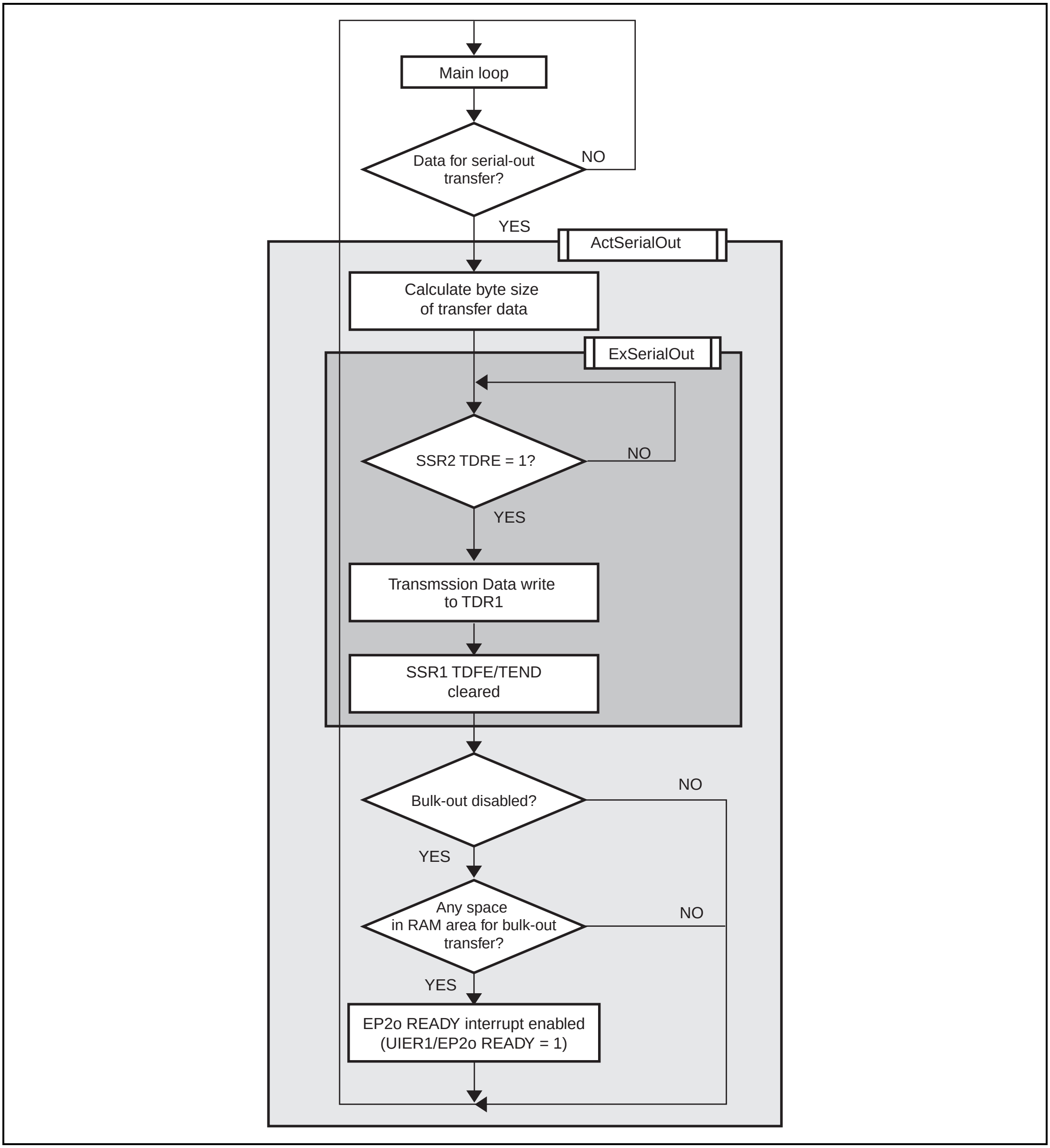


Figure 4.17 Serial-Out Transfer

4.7.2 Serial-In Transfer

Figures 4.18 and 4.19 show the operation of the sample program on serial-in transfer. When ERI1 or RXI1 reception interrupt occur, the ActSerialIn function is called.

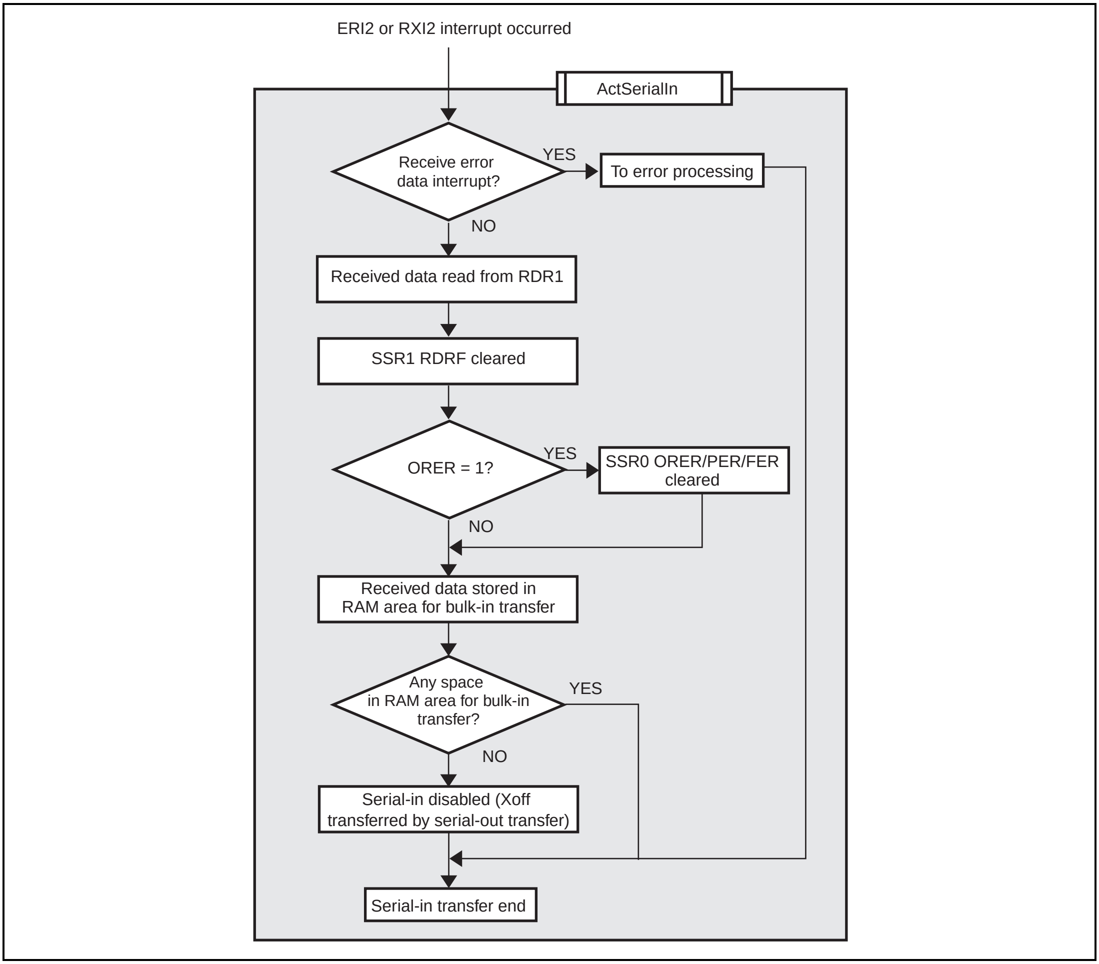


Figure 4.18 Serial-In Transfer (Receive Data Processing)

When an ERI1 interrupt which is caused by an overrun error (ORER) occurs, data is read from RDR1 in the same way as an RXI interrupt occurs. When an ERI1 interrupt which is not caused by an overrun error occurs, data in RDR1 is read to be discarded and the error flag is cleared. At this time, when a break interrupt is also received, serial reception is disabled to exit the function without clearing the FER flag. In this case, since the FER flag hold the value 1, consecutive interrupts occur and the ActSerialIn function continues to be called until a break interrupt is stopped. During these conditions, the interrupt priorities for the USB function and SCI1 modules are switched in order to enabling reception of USB interrupts.

When an overrun error occurs or data is successfully received, the data is read from RDR1 and is stored in the RAM area for bulk-in transfer. After this, the size of which the RAM area is not used is checked. When there is no area left to use, Xoff is sent to the host PC connected with serial interface in order to avoid data missing. Sending Xoff disables serial-in transfer.

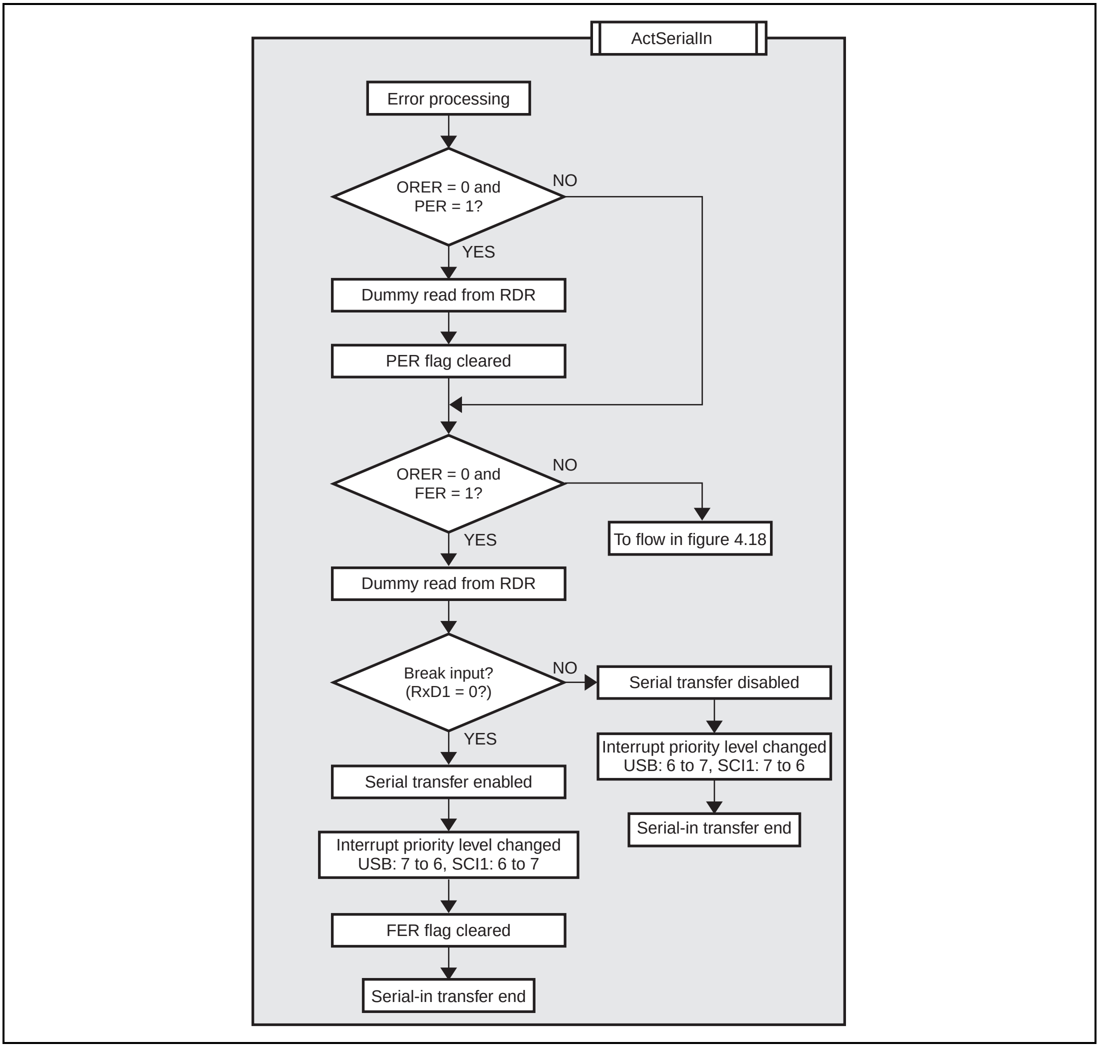


Figure 4.19 Serial-In Transfer (Error Processing)

4.8 Vendor Command

In this sample program, four vendor commands, supported by USB serial conversion driver manufactured by Hitachi ULSI Systems Co., Ltd., are decoded.

Table 4.6 shows the four vendor commands that are supported by the USB serial conversion driver.

Table 4.6(a) Vendor Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000001b	SET_LINE_CODING	Zero	Interface	8	Line Coding Structure
11000001b	GET_LINE_CODING	Zero	Interface	8	Line Coding Structure
01000001b	SET_CONTROL_LINE_STATE	Control Signal Bitmap	Interface	Zero	None
01000001b	SEND_BREAK	Duration of Break	Interface	Zero	None

Table 4.6(b) Vendor Request Code

bRequest	Value
SET_LINE_CODING	0
GET_LINE_CODING	1
SET_CONTROL_LINE_STATE	2
SEND_BREAK	3

More details of each command are explained in the following sections.

4.8.1 SetLineCoding

This request specifies parameters which are used for asynchronous data transfer.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000001b	SET_LINE_CODING	Zero	Interface	8	Line Coding Structure

Table 4.7 shows the definition of Line Coding Structure.

In this sample program, SCI1 is restarted with the settings of received Line Coding Structure on reception of this command.

Table 4.7 Line Coding Structure

Offset	Field	Size	Value	Description
0	DwDTERate	4	Number	Data terminal speed (bps)
4	BcharFormat	1	Number	Stop bit 0: 1 stop bit 1: 1.5 stop bits 2: 2 stop bits
5	BparityType	1	Number	Parity 0: None 1: Odd 2: Even 3: Mask 4: Space
6	BdataBits	1	Number	Data bits (5, 6, 7, 8)
7	BflowType	1	Number	Flow control 0: Software or none 1: Hardware

4.8.2 GetLineCoding

This request is for the host to check out the current parameter of the device. When this sample program receives this command, it returns the initial values shown in table 4.8 to the host.

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
11000001b	GET_LINE_ CODING	Zero	Interface	8	Line Coding Structure

Table 4.8 Initial Values of Line Coding Structure

Offset	Field	Size	Value	Description
0	DwDTERate	4	0x1C200	Data terminal speed (38400bps)
4	BcharFormat	1	0x0	Stop bit (1 stop bit)
5	BparityType	1	0x0	Parity (None)
6	BdataBits	1	0x8	Data bit (8)
7	BflowType	1	0x0	Flow control (Software or none)

4.8.3 SetControlLineState

This request sets the control signal.

bmRequestType	bRrequest	wValue	wIndex	wLength	Data
01000001b	SET_CONTROL_ LINE_STATE	Control Signal Bitmap	Interface	Zero	None

Table 4.9 Control Signal Bitmap

Bit Position	Description
D15 to D2	Reserved (initialized to 0)
D1	Controls transmit function of DCE 0: RTS off 1: RTS on
D0	Monitors whether or not DTE is in ready state 0: DTR off 1: DTR on

Since the H8S/2215 does not have RTS and DTR signals, only decode is carried out for this request and the DCE is not controlled.

In this sample program, it is recognized that setting the hyper terminal on the USB host PC side for communication is completed by detecting D1 = 1 and D0 = 1. At this time, a pointer that indicates the data area for bulk-in and bulk-out transfers and an internal flag in this sample program are initialized.

4.8.4 SendBreak

This request generates the break signal in device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000001b	SEND_BREAK	Duration of Break	Interface	Zero	None

The break signal transmission time (msec) is written to the wValue field. When wValue is 0xFFFF, the device continues to output the break signal until receiving the SendBreak request with wValue of 0x0000.

In this sample program, this request is decoded. A break signal, however, is not output.

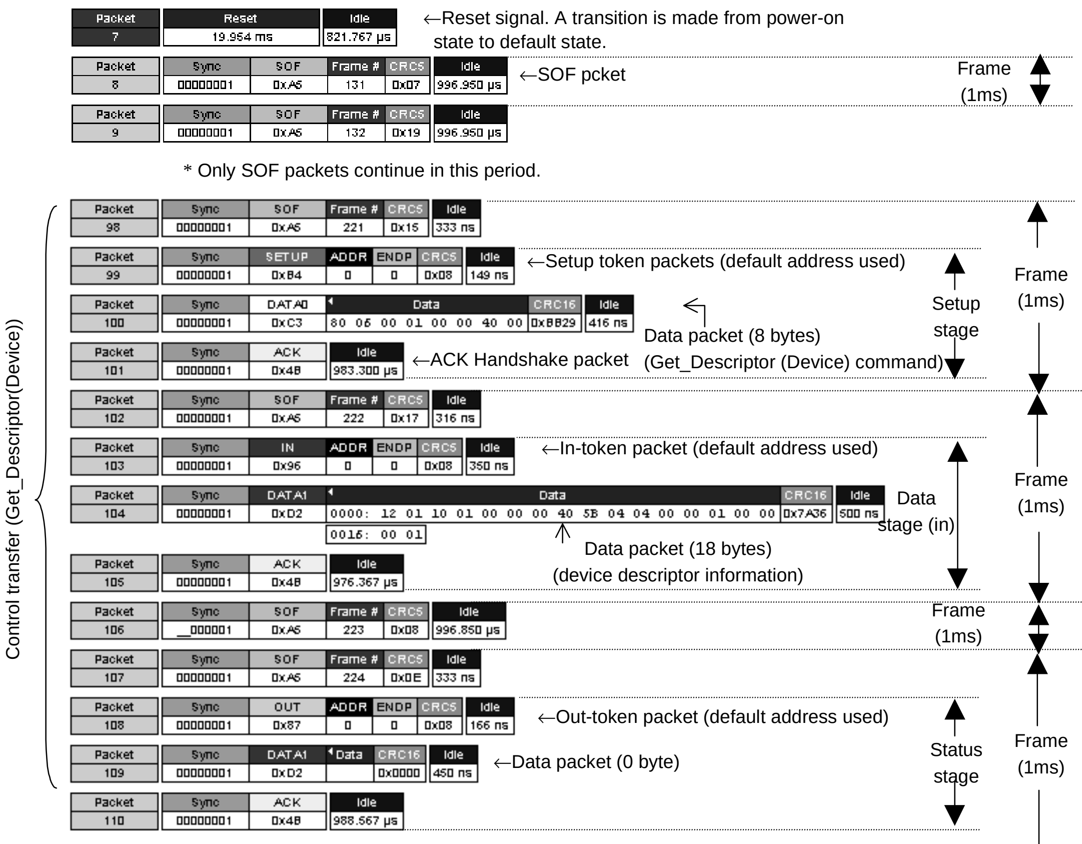
In this section, we look at how measurement is carried out with the USB Advisor, a USB protocol analyzer manufactured by CATC (http://www.catc.com), using the USB function module in the H8S/2215, and at what happens to the data as it actually flows along the bus. The following gives the description for control transfer when a device is connected and control transfer when the vendor command is transmitted as examples.

Note: The Packet # found in front of each packet is the packet number used when measuring. The Idle found at the end of each packet indicates the idle between packets.

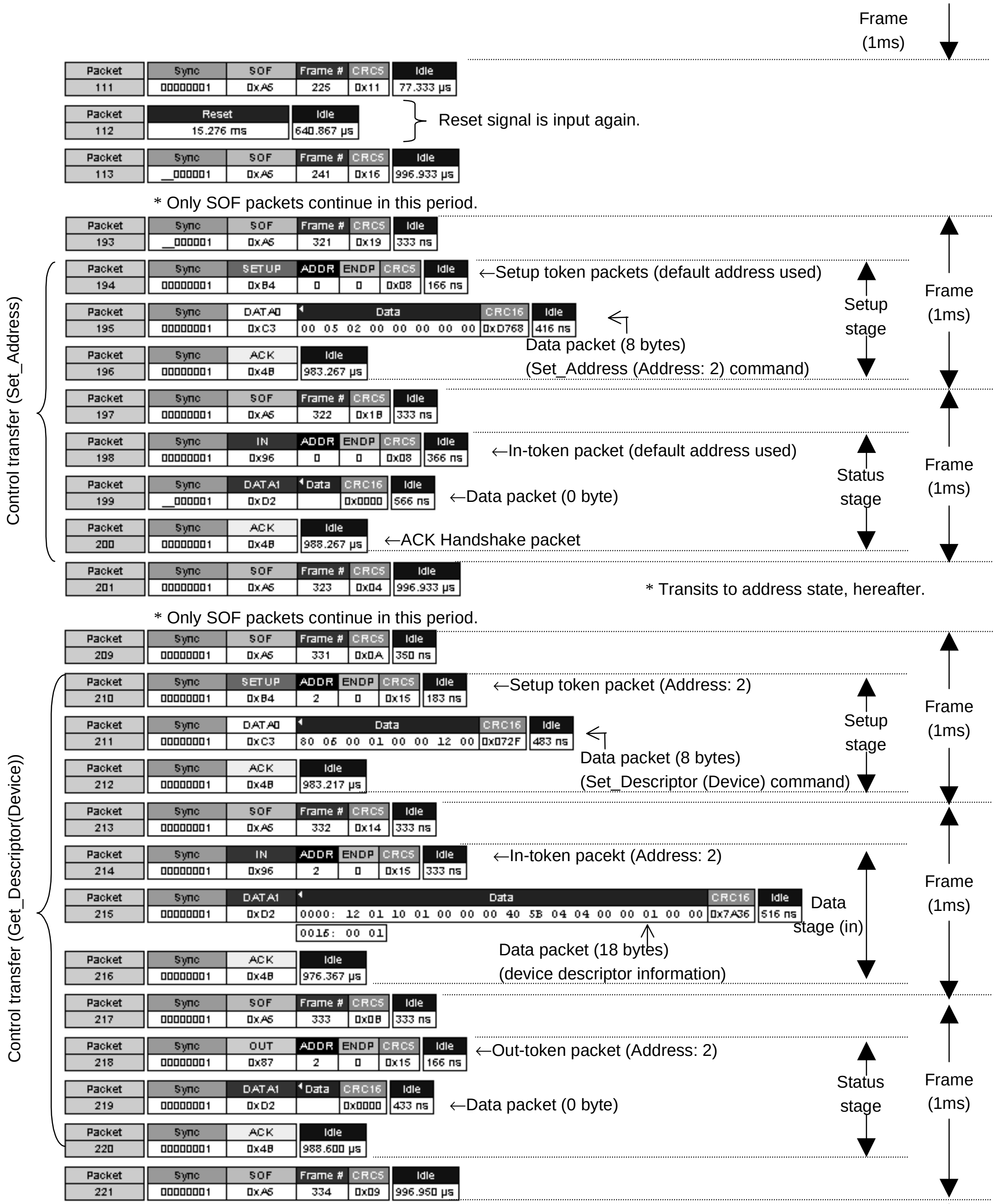
5.1 Control Transfer when Device is Connected

Figure 5.1 shows the measurement made, with a device connected to the host controller, while shifting from the power-on state (the power is supplied to Vbus) until the configuration state (device is ready for being used).

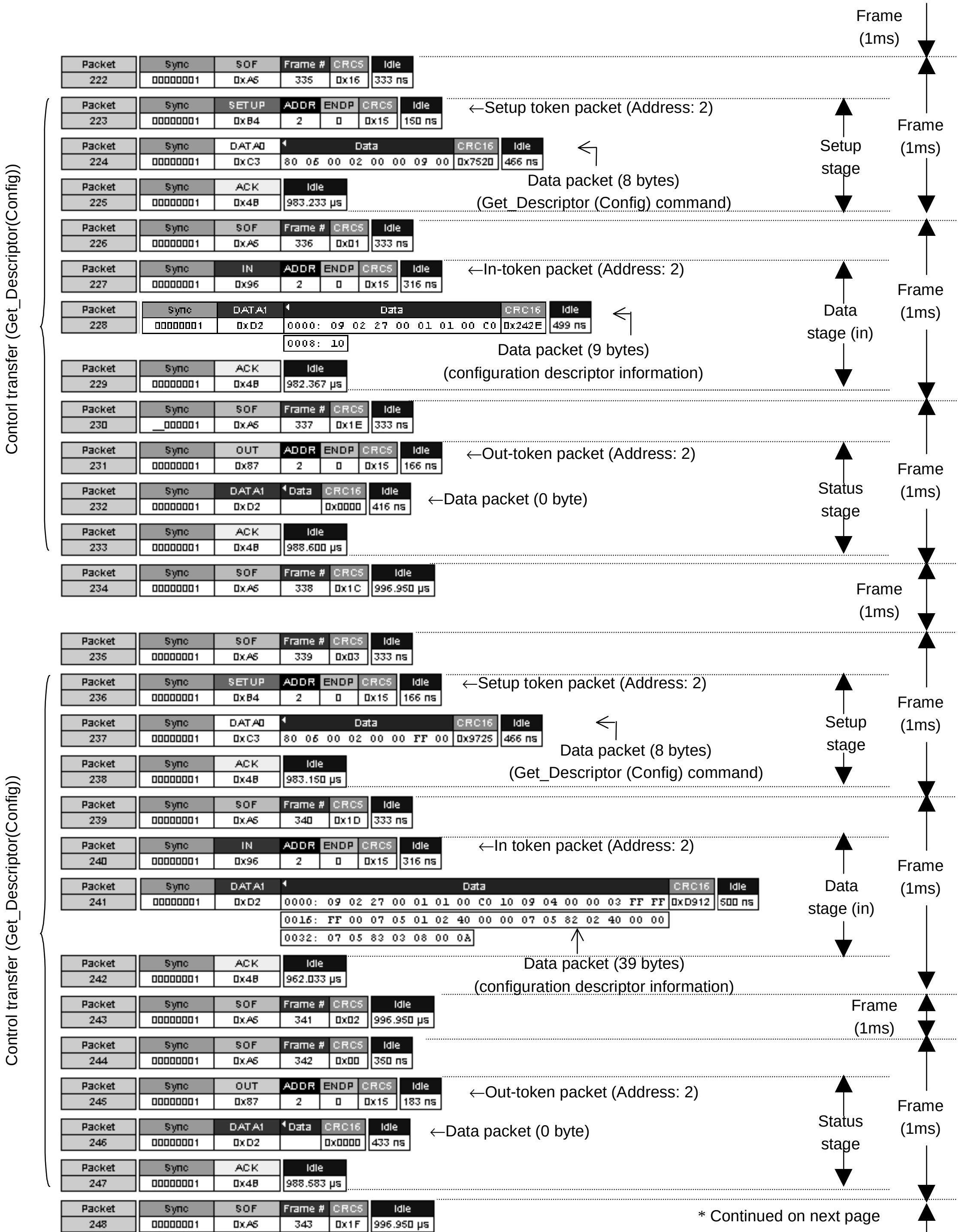
Though the packet scheduling may differ depending on the host controller, the command flow to the configuration state is always the same.



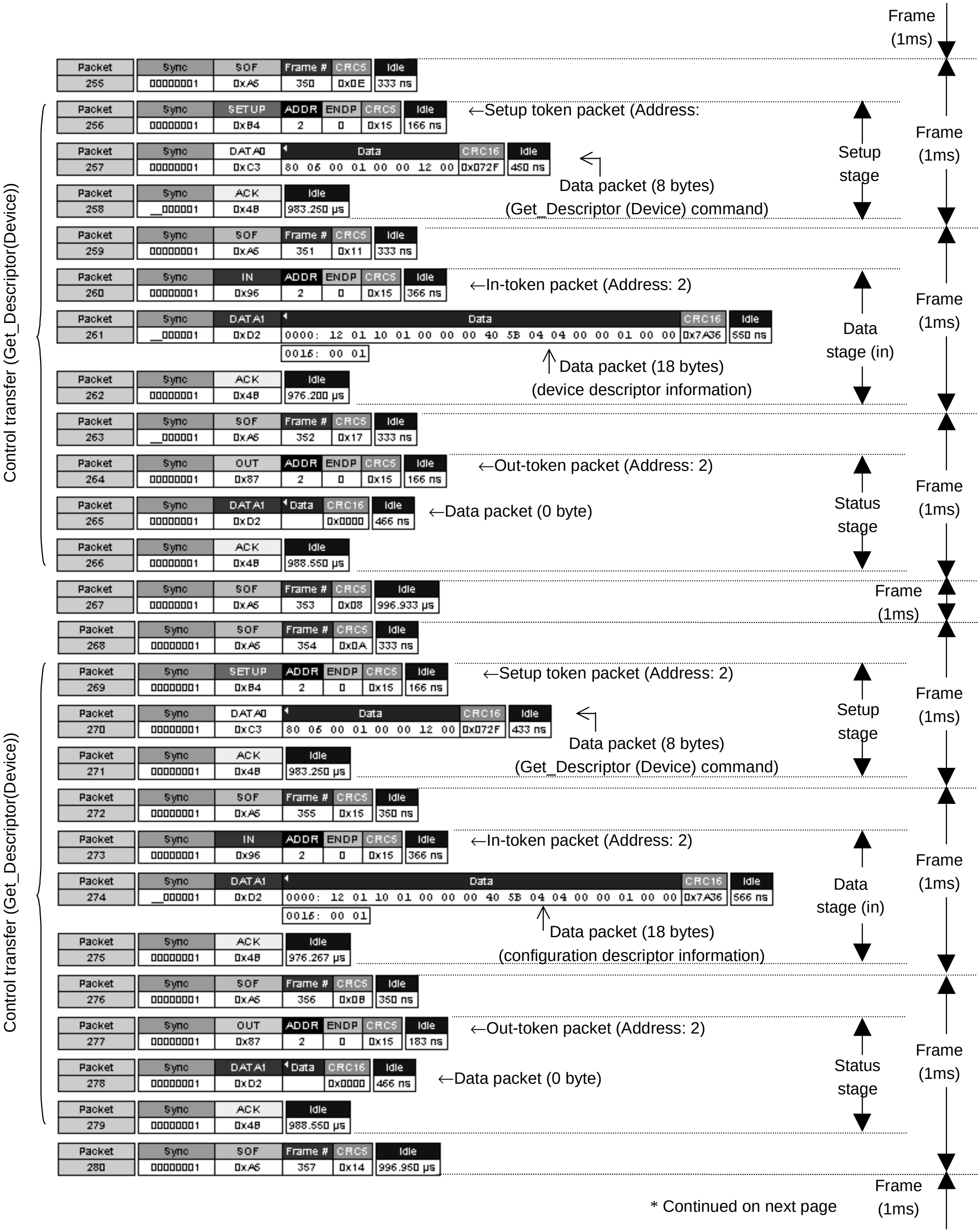
* Continued on next page



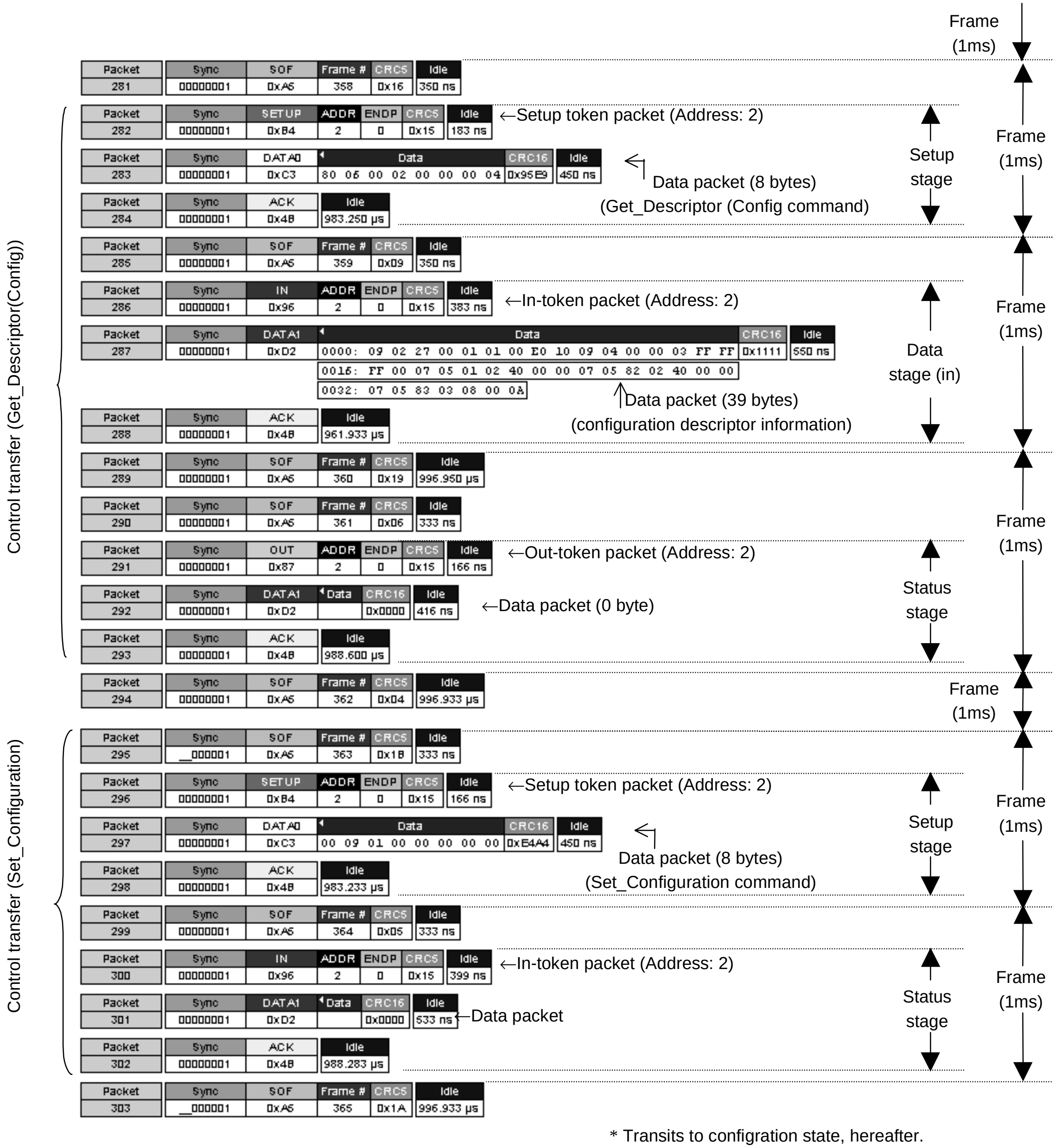
* Continued on next page



* Only SOF packets continue in this period.



* Continued on next page

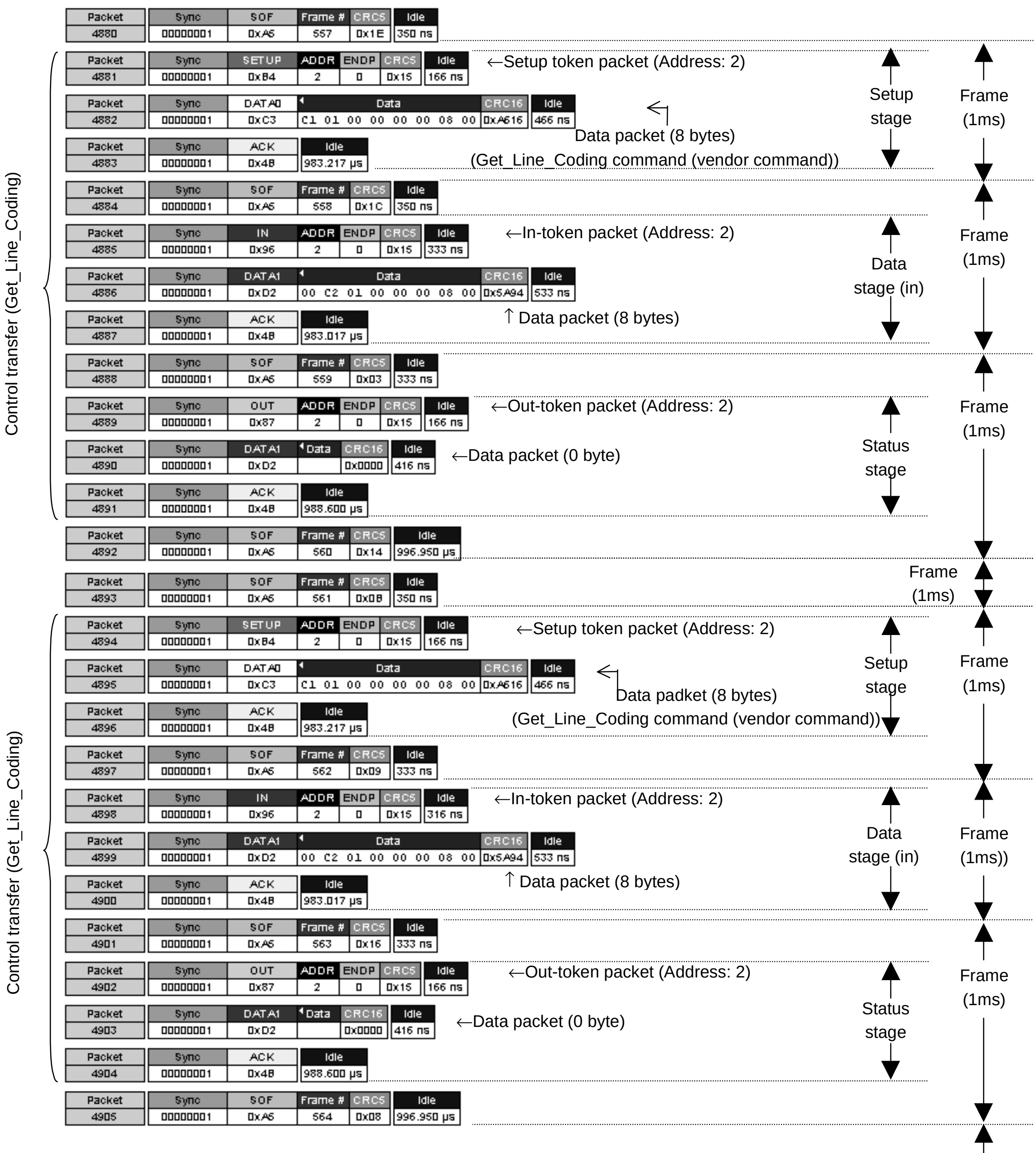


* The stationary state continues until a control transfer (vendor command) is performed.

Figure 5.1 Control Transfer when Device is Connected

5.2 Control Transfer when Vendor Command is Transmitted

Figure 5.2 shows the measurement results when the vendor command is transmitted by control transfer between the host controller and this device. (For the vendor command, refer to section 4.7.)



* Continued on next page

Control transfer (Get_Line_Coding)

Frame (1ms)

Packet	Sync	SOF	Frame #	CRC5	Idle
4906	00000001	0xA5	565	0x17	333 ns

Setup stage — Frame (1ms)

Packet	Sync	SETUP	ADDR	ENDP	CRC5	Idle
4907	00000001	0xB4	2	0	0x15	166 ns

←Setup token packet (Address: 2)

Packet	Sync	DATA0	Data	CRC16	Idle
4908	00000001	0xC3	C1 01 00 00 00 00 08 00	0xA616	466 ns

Data packet (8 bytes)
(Get_Line_Coding command (vendor command))

Packet	Sync	ACK	Idle
4909	00000001	0x4B	983.217 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4910	00000001	0xA5	566	0x15	333 ns

Data stage (in) — Frame (1ms)

Packet	Sync	IN	ADDR	ENDP	CRC5	Idle
4911	00000001	0x96	2	0	0x15	316 ns

←In-token packet (Address: 2)

Packet	Sync	DATA1	Data	CRC16	Idle
4912	00000001	0xD2	00 C2 01 00 00 00 08 00	0x5A94	533 ns

↑Data packet (8 bytes)

Packet	Sync	ACK	Idle
4913	00000001	0x4B	983.017 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4914	00000001	0xA5	567	0x0A	333 ns

Status stage — Frame (1ms)

Packet	Sync	OUT	ADDR	ENDP	CRC5	Idle
4915	00000001	0x87	2	0	0x15	166 ns

←Out-token packet (Address: 2)

Packet	Sync	DATA1	Data	CRC16	Idle
4916	00000001	0xD2		0x0000	416 ns

←Data packet (0 byte)

Packet	Sync	ACK	Idle
4917	00000001	0x4B	988.600 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4918	00000001	0xA5	568	0x1A	996.933 µs

Frame (1ms)

Packet	Sync	SOF	Frame #	CRC5	Idle
4919	00000001	0xA5	569	0x05	333 ns

Control transfer (Get_Line_Coding)

Setup stage — Frame (1ms)

Packet	Sync	SETUP	ADDR	ENDP	CRC5	Idle
4920	00000001	0xB4	2	0	0x15	166 ns

←Setup token packet (Address: 2)

Packet	Sync	DATA0	Data	CRC16	Idle
4921	00000001	0xC3	C1 01 00 00 00 00 08 00	0xA616	466 ns

Data packet (8 bytes)
(Get_Line_Coding command (vendor command))

Packet	Sync	ACK	Idle
4922	00000001	0x4B	983.217 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4923	00000001	0xA5	570	0x07	333 ns

Data stage (in) — Frame (1ms)

Packet	Sync	IN	ADDR	ENDP	CRC5	Idle
4924	00000001	0x96	2	0	0x15	316 ns

←In-token packet (Address: 2)

Packet	Sync	DATA1	Data	CRC16	Idle
4925	00000001	0xD2	00 C2 01 00 00 00 08 00	0x5A94	533 ns

↑ Data packet (8 bytes)

Packet	Sync	ACK	Idle
4926	00000001	0x4B	983.017 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4927	00000001	0xA5	571	0x18	333 ns

Status stage — Frame (1ms)

Packet	Sync	OUT	ADDR	ENDP	CRC5	Idle
4928	00000001	0x87	2	0	0x15	166 ns

←Out-token packet (Address: 2)

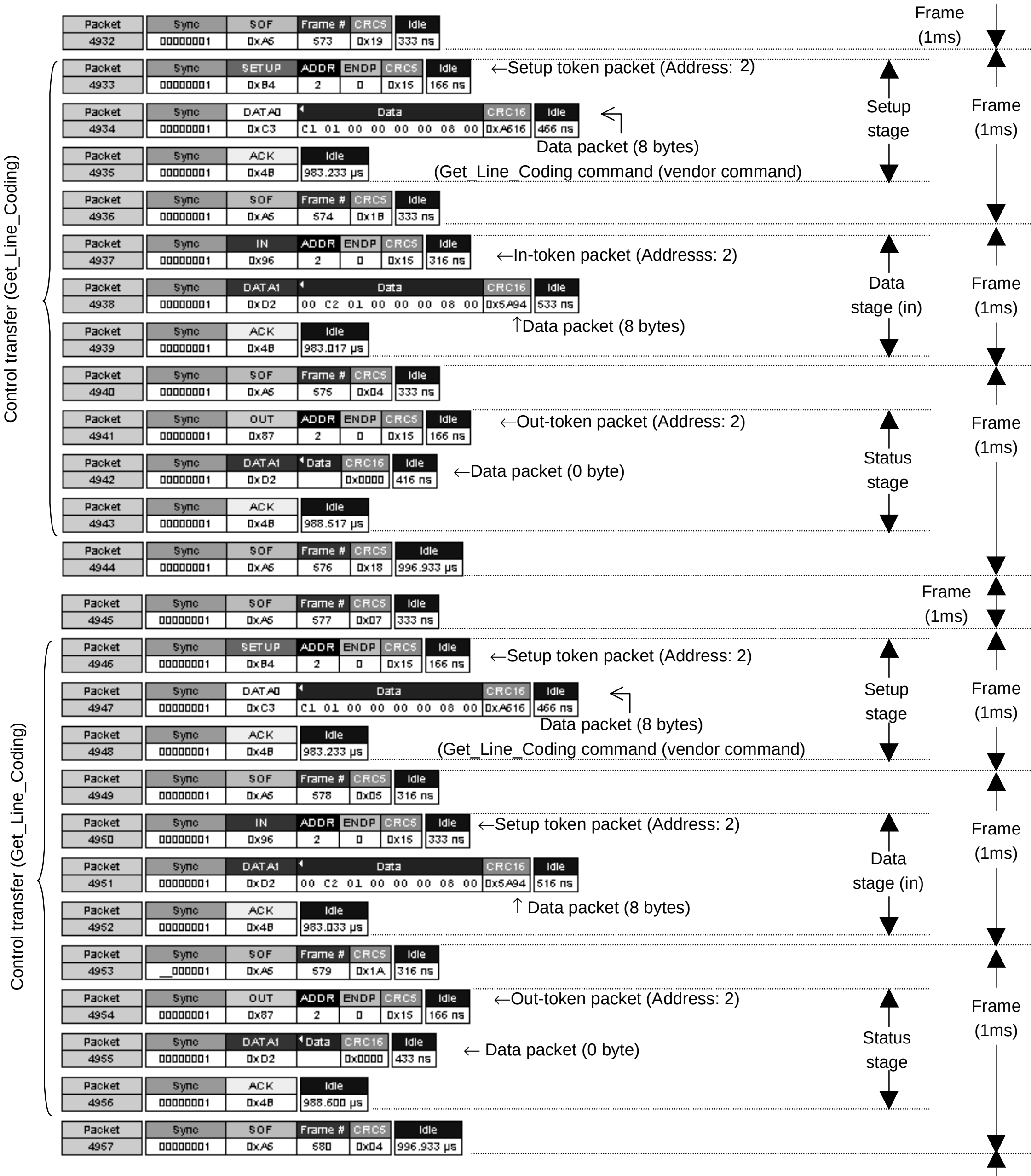
Packet	Sync	DATA1	Data	CRC16	Idle
4929	00000001	0xD2		0x0000	416 ns

←Data packet (0 byte)

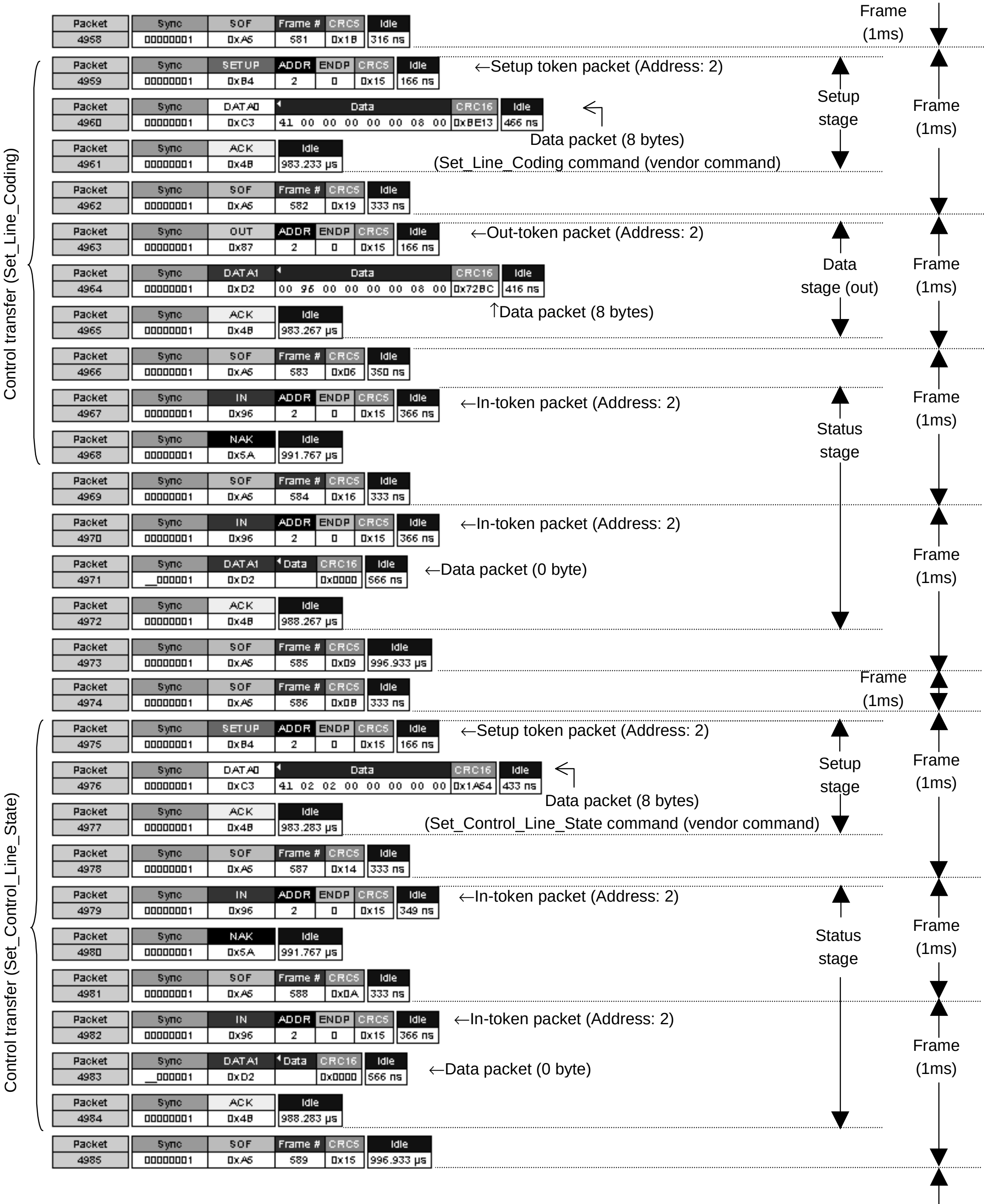
Packet	Sync	ACK	Idle
4930	00000001	0x4B	988.600 µs

Packet	Sync	SOF	Frame #	CRC5	Idle
4931	00000001	0xA5	572	0x06	996.933 µs

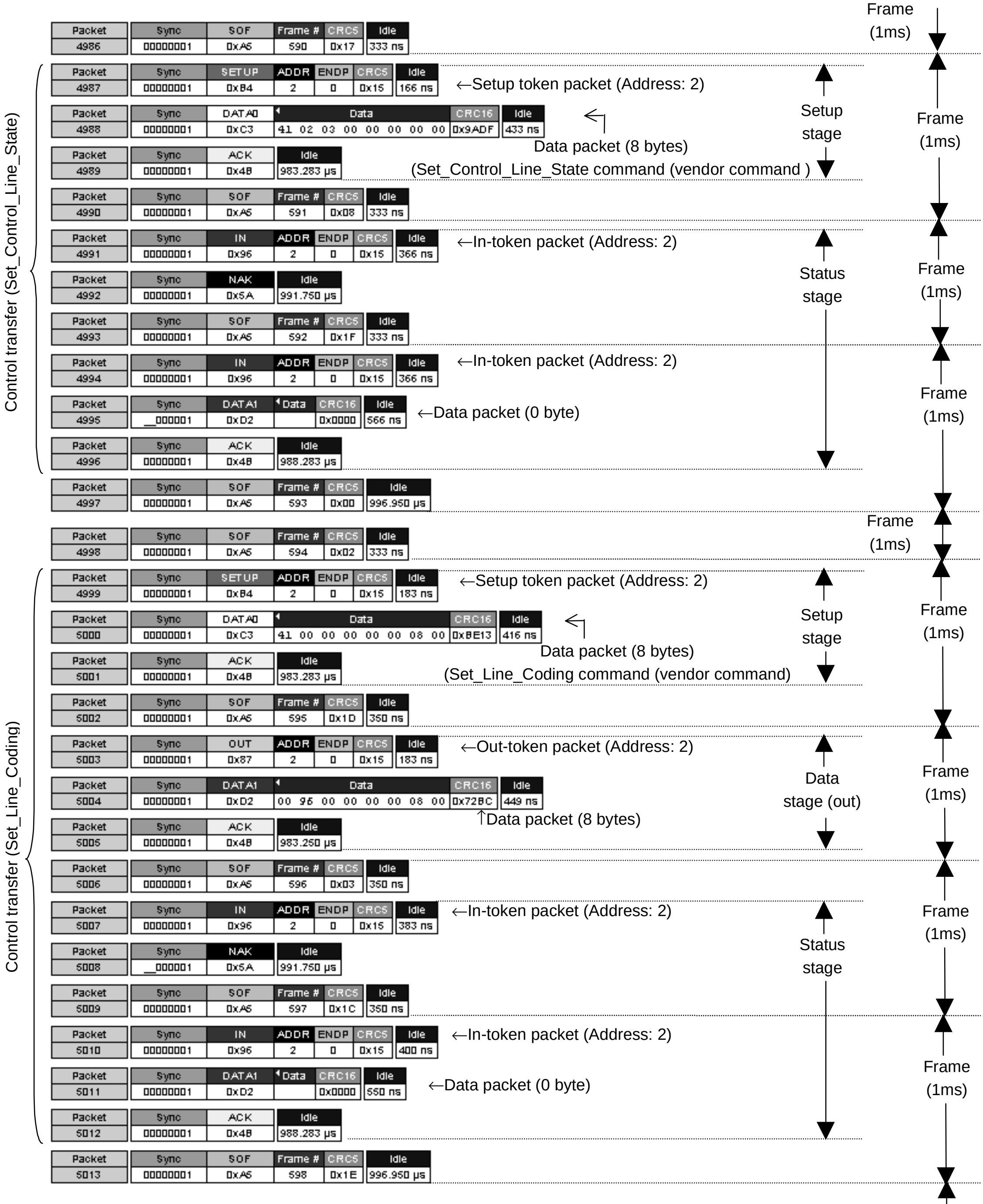
* Contrinued on next page



* Continued on next page

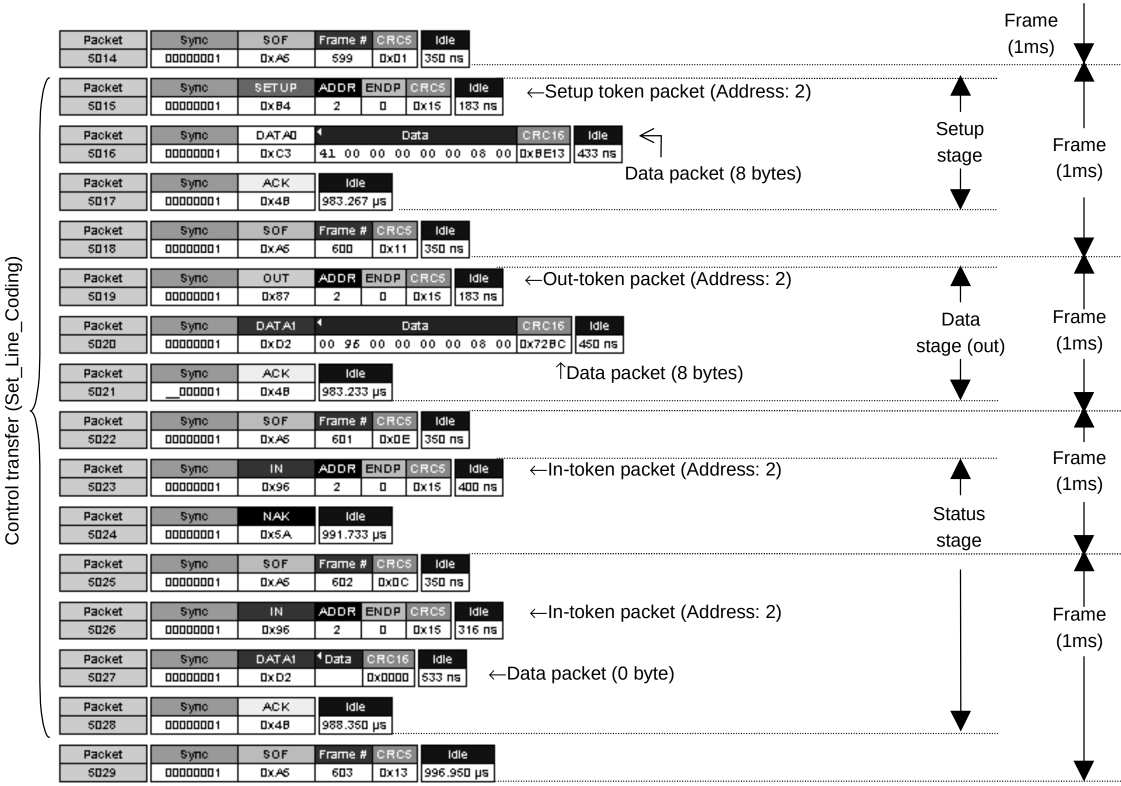


* Continued on next page



* Continued on next page

RENESAS



* The stationary state continues until a control transfer (vendor command) is performed.

Figure 5.2 Control Transfer when Vendor Command is Transmitted

H8S/2215 USB Function Module

USB Serial Conversion Application Note

Publication Date: 1st Edition, April 2002
Published by: Business Operation Division
Semiconductor & Integrated Circuits
Hitachi, Ltd.
Edited by: Technical Documentation Group
Hitachi Kodaira Semiconductor Co., Ltd.