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.

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

Cautions

Keep safety first in your circuit designs!

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

Notes regarding these materials

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

Application Notes

16

H8S/2215
USB Function Module

Application Notes

Renesas 16-Bit Single-Chip
Microcomputer

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 printer-class firmware that uses the USB Function Module in the H8S/2215. They are provided to be used as a reference when the user creates USB Function Module firmware.

Using printer-class communications as an example, the application notes describe the configuration of the USB Function Module that is built in the H8S/2215. The described system configuration is an application example of the USB Function Module, and the contents 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 1.0
- Universal Serial Bus Device Class Definition for Printing Devices
- H8S/2215 Hardware Manual
- H8S/2215 Solution Engine (MS2215CP01) Instruction Manual
- Solution Engine Single-Chip Microcomputer Based Boad (MSCCBB01) Instruction Manual
- H8S/2215 E10A Emulator User’s Manual

[Caution] The sample programs described in these application notes do not include firmware related to interrupt transfer, which is a USB transport type. When using this transfer type (see page 19-1 of the H8S/2215 Hardware Manual), the user needs to create the program for it.

Also, the hardware specifications of the H8S/2215 and H8S/2215 Solution Engine, 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 Hardware Manual and the H8S/2215 Solution Engine Instruction Manual.

Contents

Section 1 Overview 1

Section 2 Overview of the USB 3
2.1 USB Connection Topology 3
2.2 USB Signal Transfer Method 5
2.3 Recognizing a Connection vs. Non-Connection 8
2.4 USB Connector 9
2.5 Endpoint 9
2.6 USB Packets and Data Transfer 10
2.6.1 Overview of Packets 11
2.6.2 Control Transfer 15
2.6.3 Bulk Transfer 18
2.6.4 Isochronous Transfer 19
2.6.5 Interrupt Transfer 19
2.7 USB Device Framework 21
2.7.1 Device States 21
2.7.2 Device Request 22
2.8 Descriptor 24

Section 3 Development Environment 27
3.1 Hardware Environment 27
3.2 Software Environment 29
3.2.1 Sample Program 29
3.2.2 Compiling and Linking 29
3.3 Loading and Executing the Program 31
3.3.1 Loading the Program 32
3.3.2 Executing the Program 32
3.4 Printing Procedure 33

Section 4 Overview of the Sample Program 35
4.1 State Transition Diagram 35
4.2 USB Communication State 36
4.3 File Structure 37
4.4 Purposes of Functions 38

Section 5 Sample Program Operation 43
5.1 Main Loop 43
5.2 Types of Interrupts 43
5.2.1 Method of Branching to Different Transfer Processes 45

5.3 USB Operating Clock Stabilization Interrupt....................46
5.3.1 Endpoint Configuration....................47
5.4 Interrupt on Cable Connection (VBUS)....................49
5.5 Bus Reset Interrupt (BRST)....................50
5.6 Control Transfers....................50
5.6.1 Setup Stage....................51
5.6.2 Data Stage....................53
5.6.3 Status Stage....................55
5.7 Bulk Transfers....................57
5.7.1 Bulk-Out Transfers....................58
5.7.2 Bulk-in Transfers....................59

Section 6 Analyzer Data....................61
6.1 Control Transfer When a Device Is Connected....................61
6.2 Bulk-Out Transport for Printing Out (For the bulk-out transport, refer to section 2.6.3.).67

Figures

Section 1 Overview

Figure 1.1 System Configuration Example 2

Section 2 Overview of the USB

Figure 2.1 Connection Topology..3
Figure 2.2 Logical Topology..4
Figure 2.3 USB Cable Configuration (for full-speed devices) 5
Figure 2.4 NRZI Transfer Method 6
Figure 2.5 Bit Stuffing 6
Figure 2.6 SOP and SYNC...6
Figure 2.7 EOP...7
Figure 2.8 For Full-Speed Devices..8
Figure 2.9 For Low-Speed Devices..8
Figure 2.10 Type A Connector Figure 2.11 Type B Connector...9
Figure 2.12 Transactions and Frames...10
Figure 2.13 SOF Packet 12
Figure 2.14 Token Packet...12
Figure 2.15 Data Packet 13
Figure 2.16 Handshake Packet 14
Figure 2.17 Special Packet..14
Figure 2.18 Setup Stage 15
Figure 2.19 Data stage (left: IN, right: OUT)...16
Figure 2.20 Status Stage..17
Figure 2.21 Data Stage Interrupted 17
Figure 2.22 Bulk Transfer (left: IN, right: OUT) 18
Figure 2.23 Isochronous Transfer (left: IN, right: OUT) 19
Figure 2.24 (a) Interrupt-In Transfer 20
Figure 2.24 (b) Interrupt-Out Transfer 20
Figure 2.25 USB Device State 21

Section 3 Development Environment

Figure 3.1 Device Connections 27
Figure 3.2 Files Included in the Folder 29
Figure 3.3 Creating a Working Folder 29
Figure 3.4 Compile Results..30
Figure 3.5 Memory Map 31
Figure 3.6 Reset Request Dialog...32
Figure 3.7 Command Line Input...32

Section 4 Overview of the Sample Program

Figure 4.1 State Transition Diagram..35

Figure 4.2 USB Communication State 36
Figure 4.3 Interrelationship between Functions 42

Section 5 Sample Program Operation

Figure 5.1 Main Loop 43
Figure 5.2 Types of Interrupt Flags 44
Figure 5.3 USB Operating Clock Stabilization Interrupt 46
Figure 5.4 Endpoint Configuration in the Sample Program 47
Figure 5.5 Interrupt on Cable Connection 49
Figure 5.6 Bus Reset Interrupt 50
Figure 5.7 Control Transfers 50
Figure 5.8 Status in Control Transfers 51
Figure 5.9 Setup Stage 52
Figure 5.10 Data Stage (Control-In Transfer) 53
Figure 5.11 Data Stage (Control-Out Transfer) 54
Figure 5.12 Status Stage (Control-In Transfer) 55
Figure 5.13 Status Stage (Control-Out Transfer) 56
Figure 5.14 Bulk Transfers 57
Figure 5.15 Bulk-Out Transfers 58
Figure 5.16 Bulk-In Transfers 59

Section 6 Analyzer Data

Figure 6.1 Control Transfer When a Device is Connected 66
Figure 6.2 Bulk-Out Transport for Printing Out 67

Tables

Section 2 Overview of the USB

Table 2.1 Relationship between Signal Lines and Connected Devices 8
Table 2.2 Number of Available Endpoints 9
Table 2.3 Max. data size (in bytes) 10
Table 2.4 List of PIDs 11
Table 2.5 List of Standard Commands 22
Table 2.6 Device Descriptor 24
Table 2.7 Configuration Descriptor 25
Table 2.8 Interface Descriptor 25
Table 2.9 Endpoint Descriptor 26

Section 3 Development Environment

Table 3.1 Jumper Settings 28

Section 4 Overview of the Sample Program

Table 4.1 File Structure 37
Table 4.2-1 UsbMain.c 38
Table 4.2-2 StartUp.c 39
Table 4.2-3 ppout.c 39
Table 4.2-4 DoRequest.c 40
Table 4.2-5 DoControl.c 40
Table 4.2-6 DoBulk.c 40
Table 4.2-7 DoRequestPrinterClass.c 41

Section 5 Sample Program Operation

Table 5.1 Interrupt Types and Functions Called on Branching 45
Table 5.2 Transfer Types and UEPIRs 47
Table 5.3 UEPIR Settings 48

Section 1 Overview

These application notes describe how to use the USB Function Module that is built into the H8S/2215, and contain 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 controls
- 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.
- An internal bus transceiver
- Endpoint configuration selectable

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 Name	Name	Transfer Type	Max. Packet Size	FIFO Buffer Capacity	DMA Transfer
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.

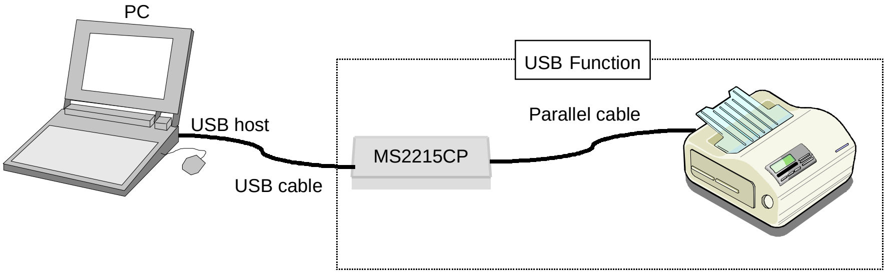


Figure 1.1 System Configuration Example

This system is configured of the H8S/2215 Solution Engine made by Hitachi ULSI Systems Co., Ltd. (hereafter referred to as the MS2215CP), a printer with a parallel port, and a PC containing Windows 2000 operating system.

The system can receive print data, transmitted from a host PC to the USB, by means of the MS2215CP, and after converting them into the parallel format, can output the print data to a printer. In addition, the system can use USB printer-class device drivers that are standard items in Windows 2000, as well as printer device drivers.

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 can be used, enabling efficient debugging.
4. Additional programs can be created to support interrupt transfer and isochronous transfer. *

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

Section 2 Overview of the USB

This chapter describes USB standards, including connection topology, transfer methods, and data formats, for your reference in developing USB systems. For details on these standards, refer to Universal Serial Bus Specification Revision 1.0.

2.1 USB Connection Topology

Figure 2.1 shows USB connection topology. A USB comprises a Host Controller mounted on a PC and devices that are connected to the Host Controller. By using a special device called a hub, you can expand the bus in order to increase the number of devices that can be connected to it. A particular type of hub, one that is directly connected to the Host Controller, is called the root hub, which is normally housed in the PC system unit. A maximum of five levels of hubs (except for the root hub) can be connected (or five hubs when connected serially).

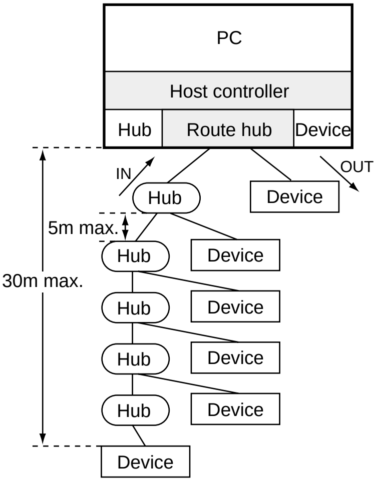


Figure 2.1 Connection Topology

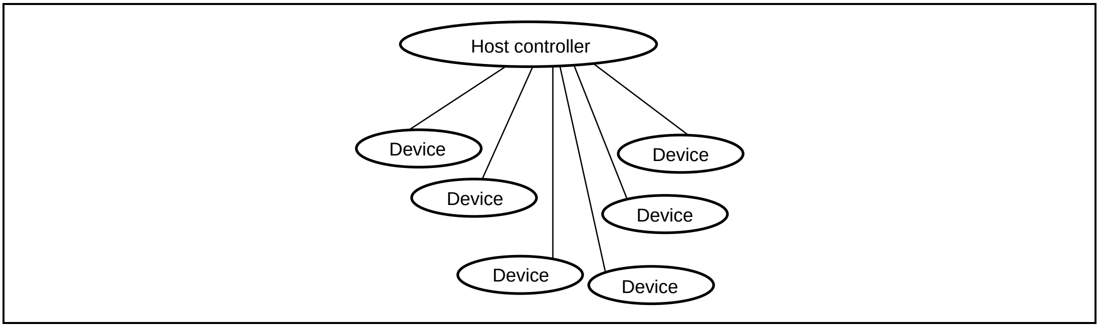


Figure 2.2 Logical Topology

The Host Controller keeps track of devices by assigning 7-bit addresses to them. Because a temporary address (default address: 0000000b) is needed that is used after a device is connected until an address is assigned to it, the maximum number of devices, including the hubs, that can be connected to the Host Controller is 127.

The actual connection topology takes the Tree form, shown in figure 2.1; however, the logical topology will be the Star form, illustrated in figure 2.2, a form in which the Host Controller and the devices perform one-to-one communications in a time division protocol. All time-division schedules (even when a device is connected via a hub, it acts as an image that is directly linked to the Host Controller) are decided by the Host Controller. Therefore, unless a command is issued by the Host Controller (for details, see Token Packets in section 2.6.1), a device never sends data to the Host Controller.

Devices can operate in two transfer modes: full speed device mode that performs high-speed transfers (12 Mbps), and low-speed device mode that performs slow transfers (1.5 Mbps).

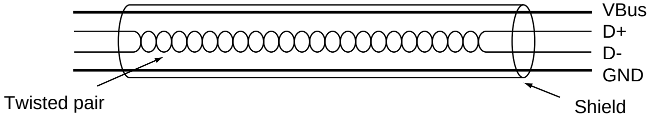
The direction in which a data transfer takes place is defined from the point of view of the Host Controller: the direction in which data flow from the Host Controller to a device is designated the OUT direction; the direction in which data flow from a device to the Host Controller is designated the IN direction.

In the OUT direction, data are transferred in a broadcast mode, wherein they are transferred to all devices that are connected. Only data with a speed of 1.5 Mbps are transferred to low-speed devices. (12 Mbps data are filtered by either the root hub or regular hubs. For further details see Special Packets in section 2.6.1.)

Token packets that are transmitted in the broadcasting OUT direction contain address information (see Token Packets in section 2.6.1 for details) that enables the devices to identify the data being sent. Based on the address information, only the device to which the address applies operates and responds to the data.

2.2 USB Signal Transfer Method

The USB comprises two signal lines (D+, D-) and two power lines (Vbus, GND). Matching this organization, the USB cable is also internally comprised of four lines as illustrated in figure 2.3. In cables for full-speed devices, the signal lines (D+, D-) have a twisted pair structure. Although full-speed device cables require shielding in addition to twisted pairs, cables used for low-speed devices require neither twisted pairs nor shielding. The maximum cable length supported is 5 m for full-speed devices and 3 m for low-speed devices, for which neither twisted pairs nor shielding is required.



Note: Neither twisted pairs nor shielding are required in low-speed device cables.

Figure 2.3 USB Cable Configuration (for full-speed devices)

Data are transferred by means of differential signals using D+, D-. The transfer method employed is the Non-Return to Zero Invert (NRZI) method, illustrated in figure 2.4, wherein when the source data are 0, D+ and D- invert, and when they are 1, no inversion occurs. In NRZ, the occurrence of successive 1s in the source data results in a lack of signal changes, which creates the potential problem of a shift in synchronization between host and device. To prevent this problem, when successive 1s occur in 6 or more bits, a 0 is inserted to cause an inversion (in a process called bit stuffing). The 0s inserted in this manner are removed by the receiving device after the data are transferred.

In a state called the idle state where no data are transferred, in full-speed devices D+ becomes the high level, and D- the low level; in low-speed devices, D+ becomes the low level, and D- the high level, according to the pull-up resistance in the device.

In the USB, data are transferred in packets (see section 2.6 for details on packets).

The leading packet is called SYNC (synchronization) with a fixed value of 00000001.

The portion of a packet in which the first bit of SYNC is inverted from D+ or D-from the idle state is called a SOP (Start Of Packet) (figure 2.6).

The end of a packet is a special signal for identifying the end of the packet, where both D+ and D- are low levels (2-bit time), which is called an EOP (End Of Packet) (figure 2.7).

In the figures below, 2.4, 2.5, 2.6, and 2.7, the post NRZI differential signal waveform is for the connection of a full-speed device. For the connection of a low-speed device, D+ and D- are reversed. (Note: In the EOP, both D+ and D- assume the low level, irrespective of the transfer speed for the device.)

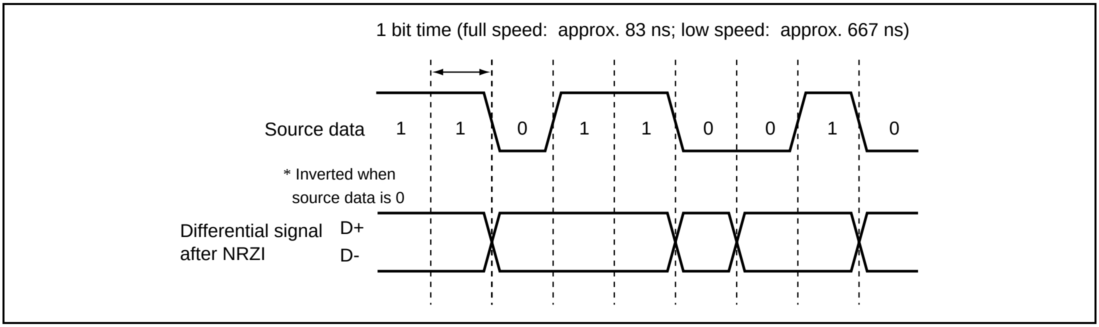

1 bit time (full speed: approx. 83 ns; low speed: approx. 667 ns)
Source data
1 1 0 1 1 0 0 1 0
* Inverted when source data is 0
Differential signal after NRZI
D+
D-

Figure 2.4 NRZI Transfer Method

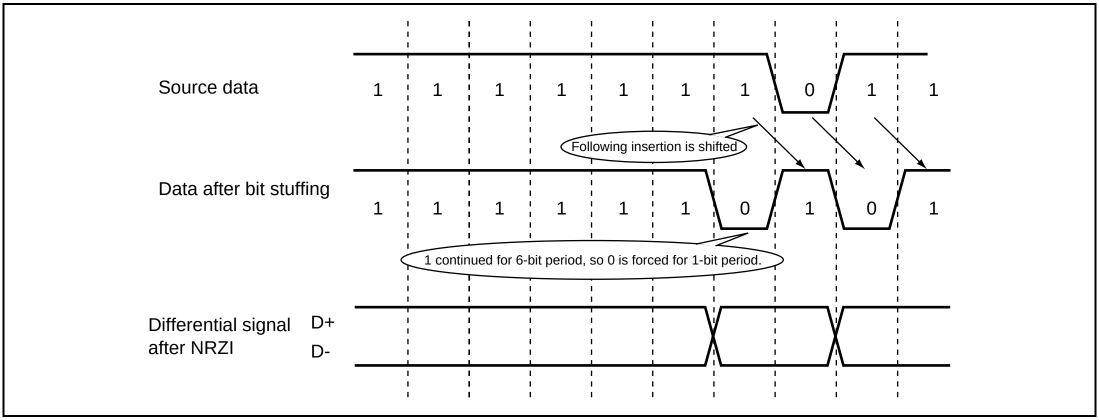

Source data
1 1 1 1 1 1 1 0 1 1
Following insertion is shifted
Data after bit stuffing
1 1 1 1 1 1 0 1 0 1
1 continued for 6-bit period, so 0 is forced for 1-bit period.
Differential signal after NRZI
D+
D-

Figure 2.5 Bit Stuffing

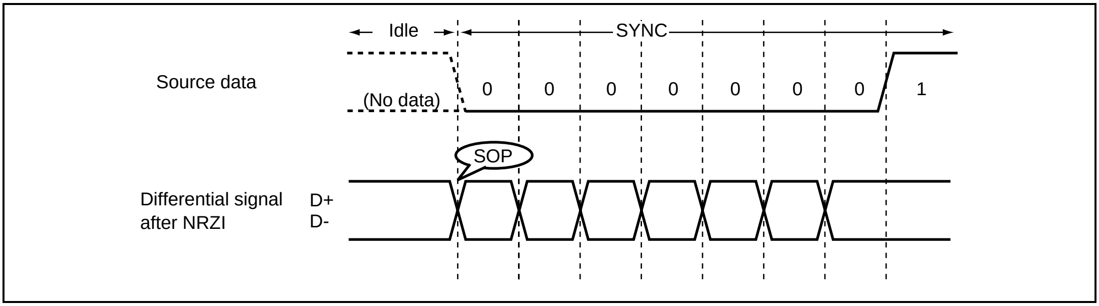


Figure 2.6 SOP and SYNC

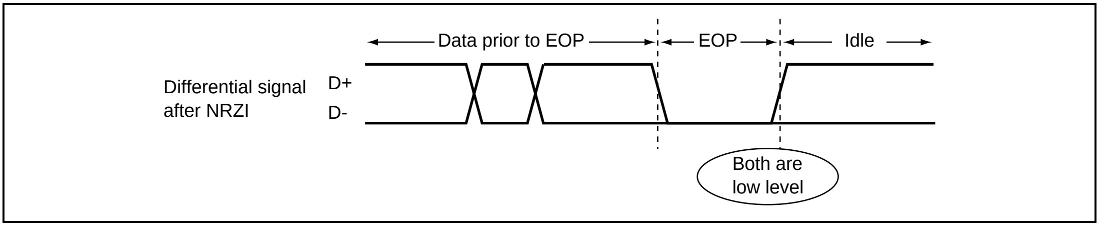


Figure 2.7 EOP

For each device, the power lines (Vbus, GND) can supply a maximum of 500 mA of current at a supply voltage of 5V.

The available current immediately after a connection is 100 mA maximum. After a connection is made, initialization is performed using a standard command (see Standard Command in section 2.7.2) using a maximum current of 100 mA.

In these settings, the Host Controller reads information on the maximum current used by devices that are connected (this information is contained in the Descriptor information to be explained in section 2.8). Based on this information, if the Host Controller determines that there are no power supply problems, the devices are allowed to increase their power consumption for the first time.

In the case of devices that require a current greater than 500 mA, a power supply must be provided in the devices themselves.

Note: If a hub that is not self-powered (a bus-powered hub) is used, the maximum current that can be used per port is subject to a 100 mA limitation. If a device requiring more than 100 mA is connected to a bus-powered hub, during the initialization process the Host Controller determines that an adequate power supply cannot be provided. In this case, the Host Controller controls the bus-powered hub so that the latter will not supply power to any of the devices that are connected to it.

2.3 Recognizing a Connection vs. Non-Connection

The side downstream from the Host Controller and the hub (the device side) pulls down the D+ and D- at 15KΩ. On the other hand, the device side pulls up the D+ for full-speed devices and the D- for low-speed devices at 1.5KΩ. Consequently, when a device is connected to the Host Controller or a hub, the Host Controller or the hub can recognize the transfer rate of the device according to which signal line, D+ or D-, is pulled up. Table 2.1 shows the relationship between the states of D+ and D- for the Host Controller/hub. Figures 2.8 and 2.9 illustrate actual circuit configurations.

Table 2.1 Relationship between Signal Lines and Connected Devices

D+	D-	Connected Device
Pulled up	Pulled down	Full-speed device
Pulled down	Pulled up	Low-speed device
Pulled down	Pulled down	Device not connected
Pulled up	Pulled up	Disabled

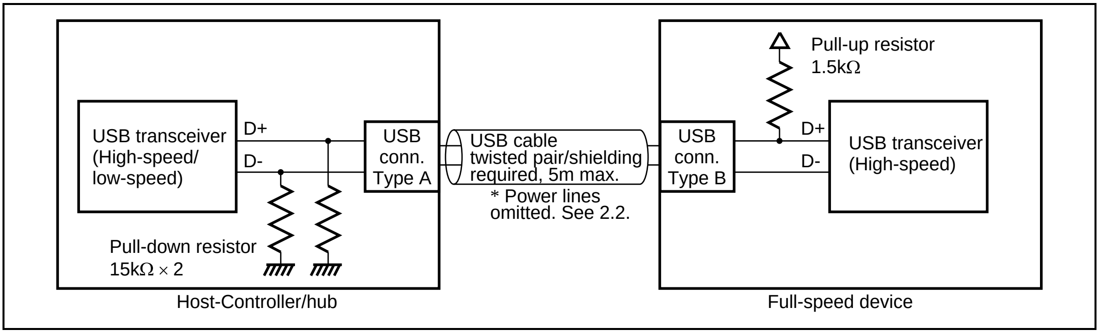


Figure 2.8 For Full-Speed Devices

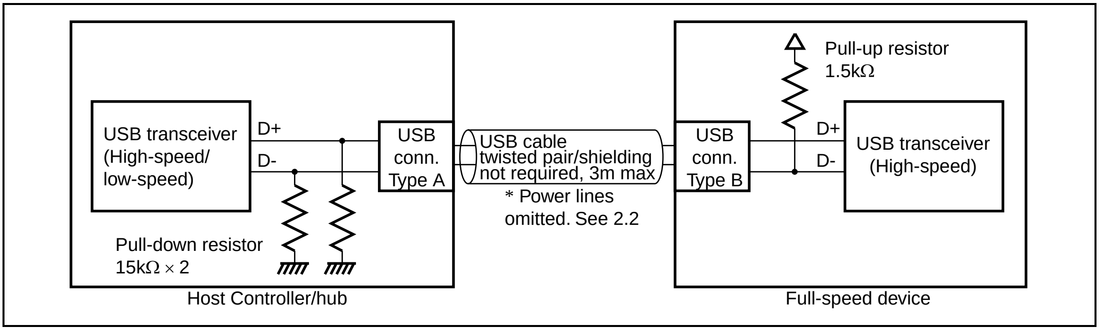


Figure 2.9 For Low-Speed Devices

2.4 USB Connector

The USB uses two types of connectors: a flat Type A connector used on the Host Controller side (figure 2.10) and a square Type B connector used on the device side (figure 2.11). The different connector configurations are designed to prevent physical misconnection (in the USB, connections between Host Controllers or devices are prohibited).

In the case of a hub, a Type B connector is used on the upstream side (the Host Controller side), and a Type-A connector is used on the downstream side (the device side).

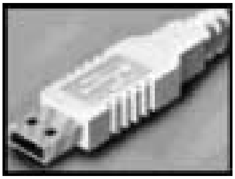


Figure 2.10 Type A Connector

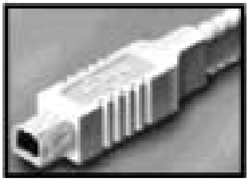


Figure 2.11 Type B Connector

2.5 Endpoint

Each device has FIFOs called endpoints (EPs). When sending or receiving data, the Host Controller and the device do so through endpoints. The number of endpoints that a device can have depends on the transfer rate for the device and is defined as in table 2.2.

Table 2.2 Number of Available Endpoints

Device Transfer Rate	End-Point No.	Max. No. of End-Points
Full speed (12 Mbps)	0 to 15	16 each for IN/OUT
Low speed (1.5 Mbps)	0 to 2	3 each for IN/OUT

In table 2.2, the endpoint with number 0 is used for control transfers (section 2.6.2). All devices must have endpoint 0. Any number of endpoints 1 ~ 15 can be used. The direction in which data flow through an endpoint or the application of an endpoint can be user-defined as part of a device design process. In USB1.0, however, interrupt transfers can occur only in the IN direction (section 2.6.5).

For endpoints, the maximum amount of data that can be sent or received is defined for each transfer method. Data greater than a specified side cannot be sent or received through a given endpoint. However, any data less than the allowed maximum size (short packets) can be sent or received. Table 2.3 shows the endpoint data sizes for each transfer method. For each endpoint, any data size within the limits defined in table 2.3 can be specified.

Table 2.3 Max. data size (in bytes)

	Transfer Method			
Device transfer rate	**Control transfer**	**Bulk transfer**	**Interrupt transfer**	**Isochronous transfer**
Full speed	8,16,32,64	8,16,32,64	0 to 64 (any integer)	0 to 1023 (any integer)
Low speed	8	Not available	0 to 8 (any integer)	Not available

Note: See sections 2.6.2 to 2.6.5 for transfer methods.

2.6 USB Packets and Data Transfer

In the USB, data are transferred in units of packets. A packet is the smallest unit of data in USB data. The USB protocol communicates using a combination of several packets, and this combination is referred to as a transaction. In a transaction, packets appear in the following order: token, data, and handshake.

A set of transactions is referred to as a frame (figure 2.12).

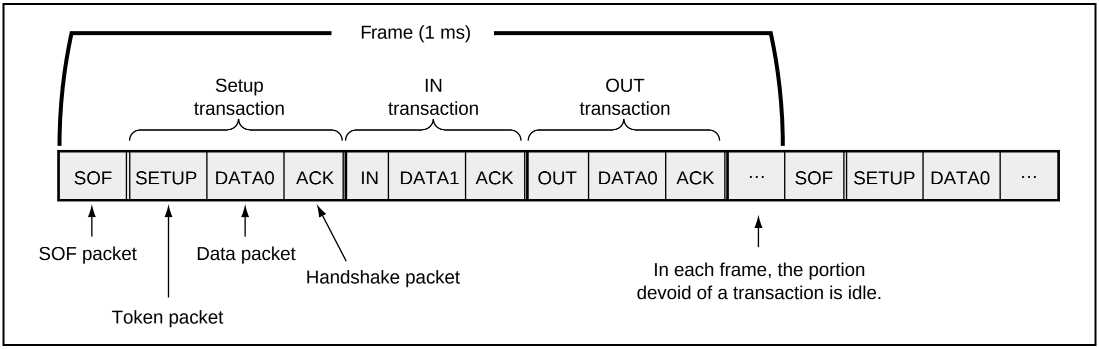


Figure 2.12 Transactions and Frames

A frame begins with an SOF packet that is issued every millisecond and continues on to the next SOF. The scheduling of transactions in a frame is handled completely by the Host Controller.

In each frame, the portion that is not filled with a transaction (the portion devoid of any data) assumes an idle state, as explained in section 2.2.

Transactions are sent and received between the Host Controller and a device according to a specified sequence. Following is a description of packets used in a USB, as well as the characteristics and the format of each transfer method.

2.6.1 Overview of Packets

Packets used in the USB must conform to prescribed formats. As shown in table 2.4, packets can be classified into five categories: SOF, token, data, handshake, and special. These categories are identified using a 4-bit PID (packet ID).

Table 2.4 List of PIDs

PID Type	PID Name	Send Device	PID[3:0]
SOF	SOF	Host controller	0101
Token	OUT	Host controller	0001
	IN	Host controller	1001
	SETUP	Host controller	1101
Data	DATA0	Host controller/device	0011
	DATA1	Host controller/device	0010
Handshake	ACK	Host controller/device	0010
	NAK	Device	1010
	STALL	Device	1110
Special	PRE	Host controller	1100

A packet takes the following format: a packet begins with SYNC, followed by PID, $\overline{\text{PID}}$, and CRC (the handshake or special packet does not have a CRC), and ends with an EOP. SYNC (synchronization) indicates the beginning of a packet and transmits a fixed value of 00000001. The receiver of the packet performs a synchronization by using SYNC. PID indicates the type of packet, and each type has a unique value. $\overline{\text{PID}}$ is a bit-by-bit binary complement of PID. This complement permits the detection of errors. CRC (Cyclic Redundancy Check) is the result of CRC-checking of each packet with the exception of SYNC, PID, and $\overline{\text{PID}}$.

SOF (Start Of Frame)

An SOF is a packet that is issued by the Host Controller at millisecond intervals. The interval from on SOF to another is called a frame. SOFs are used to synchronize an entire device. In addition, they are used to generate reference signals for isochronous transmissions (section 2.6.4) or suspend-prevention signals (generated by the hub/root hub upon receipt of a keep-alive signal: SOF) for low-speed devices. Although in terms of classification an SOF belongs to the token packet, because it is used differently from other tokens as described above, it represents a separate category.

SYNC	PID	$\overline{\text{PID}}$	Frame no.	CRC	EOP
8 bits	4 bits	4 bits	11 bits	5 bits	2 bits

PID type
SOF=0101

Figure 2.13   SOF Packet

Token

A token, which can only be issued by the Host Controller, is used to inform a device that a command is being sent or the direction in which data are to be sent. Several types of token packets exist, as described below. A token packet also includes address information that enables a given device whether data being sent from the Host Controller are addressed to it, and end-point information that identifies the endpoint for a device.

[OUT token]

The Host Controller issues an OUT token before sending data to a device.

[IN token]

The Host Controller issues an IN token when requesting the transmission of data from a device.

[SETUP token]

This token is issued when a command is transmitted in a control transfer. See section 2.6.2 for details on control transfers.

SYNC	PID	$\overline{\text{PID}}$	ADDR	ENDP	CRC	EOP
8 bits	4 bits	4 bits	7 bits	4 bits	5 bits	2 bits

PID type
OUT=0001
IN=1001
SETUP=1101

Figure 2.14   Token Packet

Data

The Host Controller and devices use the data packet when transmitting data. Two types of data packets exist, differentiated by whether PID is DATA0 or DATA1. Transmission of these data packets in an alternating fashion can detect any missing data, which enhances the reliability of the transmission process. (Isochronous transmissions use data packets that are fixed at DATA0.)

SYNC	PID	$\overline{\text{PID}}$	DATA	CRC	EOP
8 bits	4 bits	4 bits	0~1023 bytes	16 bits	2 bits

PID type
DATA0=0011
DATA1=1011

Figure 2.15 Data Packet

Handshake

A handshake enables the receiver to notify the sender of whether the data have been received normally. The following types of handshake exist: (Note: A handshake is not issued in an isochronous transfer.)

[ACK]

This handshake is issued when either the Host Controller or a device has received a data packet normally.

[NAK]

A NAK is issued by a device to the Host Controller under the following conditions:

- Although OUT token packets and data packets were received from the Host, data cannot be received because the endpoint is full.
- Although an IN token packet was received from the Host, the data to be sent are not yet ready.

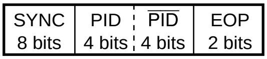
When receiving NAK, in the case of an OUT transaction, the Host Controller re-issues an OUT token and the data that failed to be received; in the case of an IN transaction, the Host Controller re-issues an IN token later. Because the Host Controller is defined as being able to send and receive data packets at any time, the Host Controller never returns NAK to a device.

[STALL]

A STALL handshake is issued by a device when an error condition occurs and the device requires intervention by the Host.

[No response] (no handshake packets issued)

If an error is found in a PID or a CRC result does not match, a handshaking is not performed, and no response is generated. If a no response condition lasts more than a fixed length of time (16~18 bit time) after transmitting data, the Host Controller or a device goes into a timeout state and recognizes that a communication error has occurred. Subsequently, the Host Controller re-issues the token and data for which an error condition was recognized.



PID type
ACK=0010
NAK=1010
STALL=1110

Note: Packets not issued if no response

Figure 2.16   Handshake Packet

Special

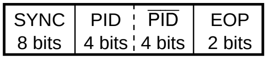
A PRE(PREAMBLE) packet is defined as a special packet. The PRE packet indicates to the device that a low-speed transfer will be performed following it.

A full-speed data transfer to low-speed device can cause an error.

The PRE packet can prevent this error.

When dealing with a low-speed device, hubs (including the root hub) filter out any full-speed data so that they are not transmitted to the low-speed device. However, when receiving a PRE packet, the hubs stop filtering, and begin to transfer the low-speed data received from the Host Controller to the low-speed device.

Although low-speed data are also transferred to full-speed devices, because low-speed data cannot generate valid full-speed PIDs, there is no possibility of full-speed devices producing an error due to the low-speed data.



PID type
PRE=1100

Note: Low-speed data following this packet

Figure 2.17   Special Packet

2.6.2 Control Transfer

A control transfer is used to issue a command to a device. This is the first transfer that occurs when a device is connected to the Host Controller. In this case, the Host Controller uses a control transfer on the new device in order to obtain information on the device. Therefore, whether they are full-speed devices or low-speed devices, all devices must support this transfer method.

Control transfers can be divided into a setup stage, a data stage, and a status stage.

Note: In the following description of transfer methods, which side sends a packet is indicated on the right side of the packet, i.e., (H) indicates the Host Controller side, (D) the device side.

[Setup Stage]

This is the first stage in a control transfer. In the setup stage, the Host Controller issues a command to a device and provides instructions on what is to be sent or received. According to this command, the device sets up the data to be sent to the Host Controller or prepares receiving data from the Host Controller.

The setup stage for a control transfer consists of setup transactions. The size of the data packet for a setup transaction is always 8 bytes. The Host Controller stores the command being sent in the data packet.

The PID for a data packet is always DATA0. The handshake packet for a setup transaction is the packet that the device sends to the host. In this case, the device must always return ACK. Returning either NAK or STALL in a setup transaction is prohibited. Therefore, devices must always be prepared to receive a setup transaction.

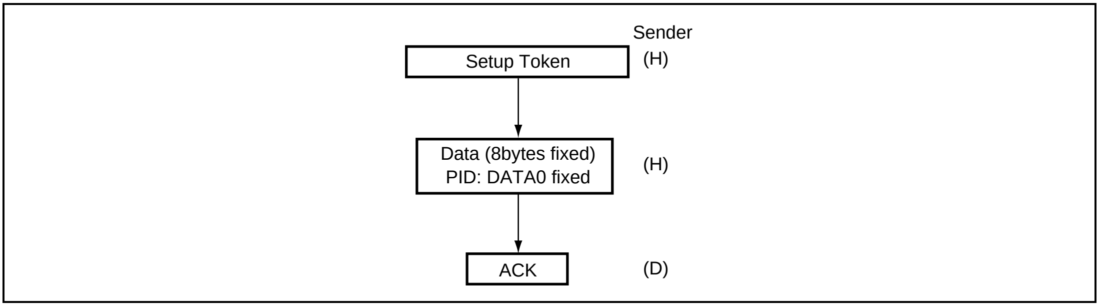


Figure 2.18 Setup Stage

[Data Stage]

In the data stage, according to the command received in the setup stage, the device repeats the receipt of the data being sent or the transmission of the data to be sent.

The direction of data never changes in the midst of a data stage.

In an IN direction data stage, if the data to be sent by the device have depleted, the device uses either a short packet (a data packet with a byte count less than the maximum data size specified for the device) or a 0-byte data packet to notify the Host Controller of the end of transmission.

Some commands do not have any data to be sent or received, in which case the data stage itself is omitted.

In cases where data are sent/received repeatedly, the PID for the data packets toggles DATA1→DATA0→DATA1...

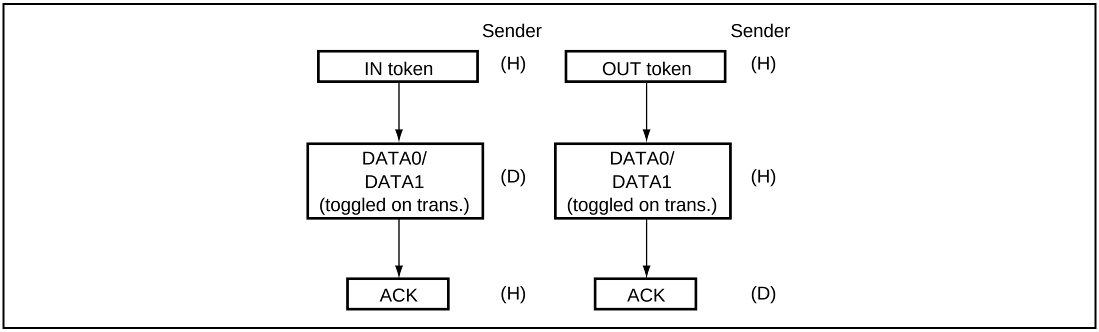


Figure 2.19 Data stage (left: IN, right: OUT)

[Status Stage]

A status stage begins when a token is transmitted in a direction opposite to the data stage (or the setup stage if there is no data stage). For example, if an IN token is issued in a data stage and data are transferred from a device to the Host Controller, the status stage begins when an OUT token is issued. Thus, the data stage terminates when the direction of data is reversed.

As illustrated in figure 2.20, a status stage is associated with three patterns: an IN direction data stage, an OUT direction data stage, and no data stage.

The data packet following the transmission of a token in the status stage must contain a packet with a 0-byte data length with a DATA1 PID.

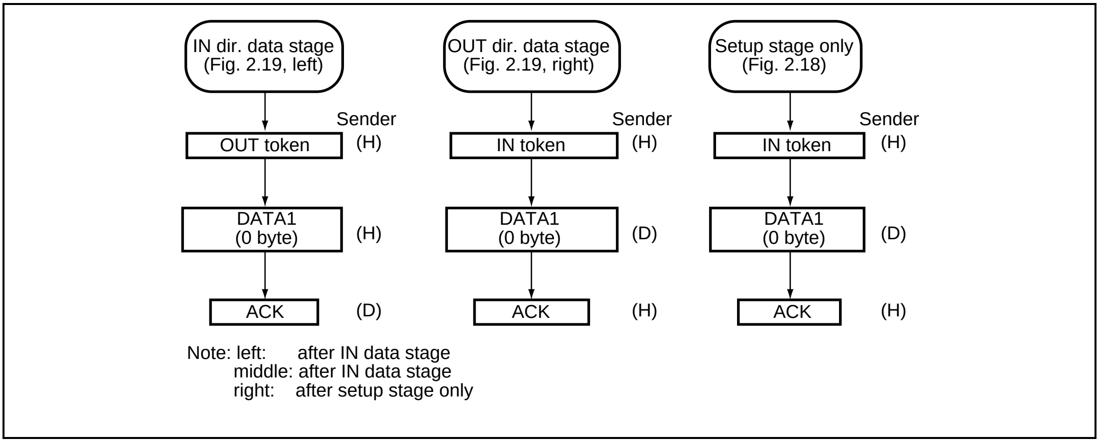


Figure 2.20 Status Stage

The reason that the reversal of direction brings on the status stage is that the data stage is defined so that it can be terminated even before the Host Controller has received or transmitted all the data that were requested by means of a setup stage command.

Figure 2.21 shows an example of a control transfer that has an IN direction data stage. Suppose that the Host Controller requests 32-byte data in the setup stage; after the setup stage has ended, the Host Controller issues an IN token; according to this command, the device sends 88-byte data (if the maximum packet size is 88 bytes); and the Host Controller issues ACK. At this point, the device will have sent 8 bytes out of the 32 bytes. If more data are needed, the Host Controller re-issues the IN token. When no more data are needed, the Host Controller issues the OUT token. The OUT token changes the direction of data, and at this time the status stage is brought on, and the control transfer ends.

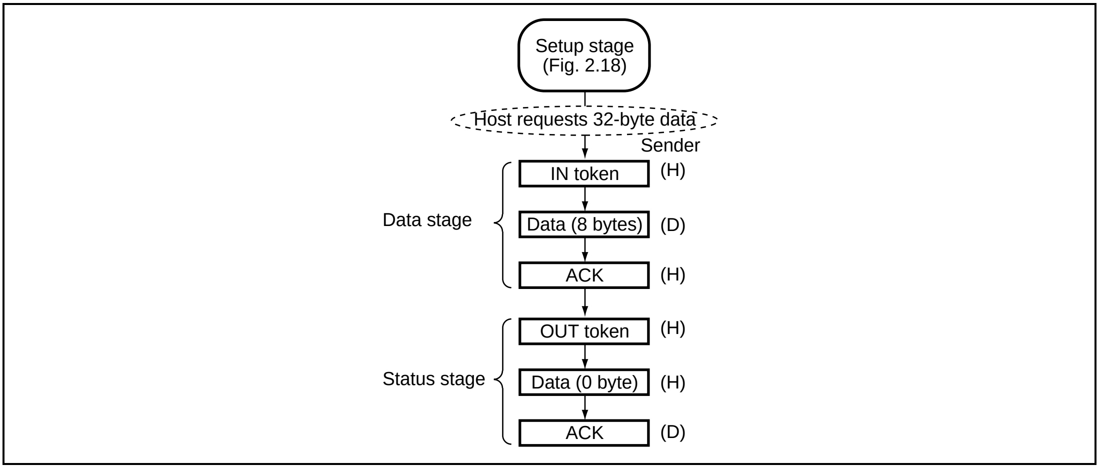


Figure 2.21 Data Stage Interrupted

2.6.3 Bulk Transfer

A bulk transfer is used to send large quantities of data without error when the transfer process is not subject to a time constraint. In a bulk transfer, the data transfer speed is not guaranteed, but data integrity is guaranteed. If a data error is found (e.g., a CRC mismatch), the receiver does not issue a handshake. If ACK is not returned, the sender re-transmits the affected data. If there is no room in the FIFO or the data to be sent are not yet ready, the sender issues NAK. The amount of data that can be transferred in a bulk transfer can be specified in the MAX packet size. A bulk transfer cannot be used with low-speed devices.

If an IN token is issued by the Host Controller (left side in figure 2.22), data are transmitted from the device and a handshake is issued by the Host Controller.

If an OUT token is issued by the Host Controller (right side in figure 2.22), data are transmitted from the Host Controller, and a handshake is issued by the device.

In both bulk IN/OUT, each time a data send/receive action is repeated, the PID for the data packet toggles DATA0→DATA1→DATA0...

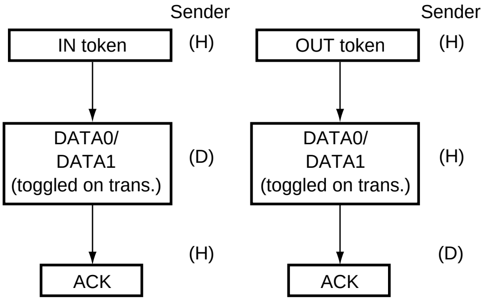


Figure 2.22   Bulk Transfer (left: IN, right: OUT)

2.6.4 Isochronous Transfer

An Isochronous transfer is used to send continuous data, such as audio data and moving pictures. Isochronous transfers are priority-scheduled so that a data transfer occurs at a rate of once per frame (1 ms). In an Isochronous transfer, however, offset values from an SOF packet cannot be guaranteed. In other words, the first transfer can occur at the end of a frame and the next transfer can occur at the beginning of the frame. Devices are required to be able to handle these contingencies.

Isochronous transfers cannot be used with low-speed devices.

As shown in figure 2.23, Isochronous transactions do not contain handshake packets.

Unlike a bulk transfer, in an Isochronous transfer, data are not re-sent even if there are errors in the data that are transferred. The maximum size of a data packet that can be specified for an Isochronous transfer is 1023 bytes.

The PID for the data packet is fixed at DATA0 (the PID does not toggle).

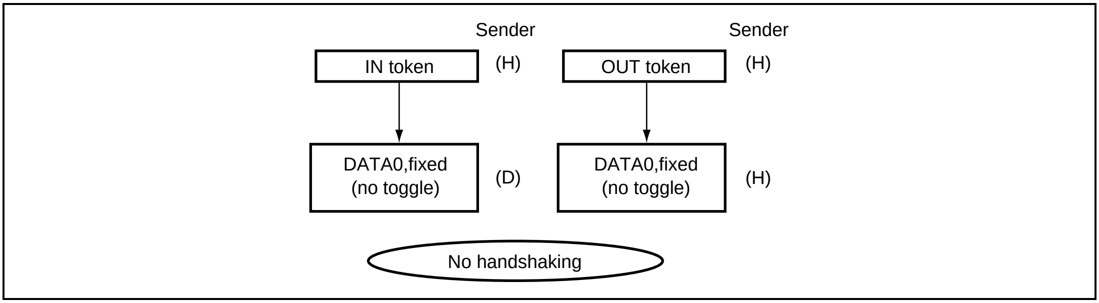


Figure 2.23 Isochronous Transfer (left: IN, right: OUT)

2.6.5 Interrupt Transfer

In an interrupt transfer, the Host Controller generates IN transactions for devices in specified cycles. Devices can specify to the Host Controller the cycle in which transactions are to be generated. A cycle can be specified in 1 to 255 frames. The Host Controller starts an IN transaction at least once per specified cycle. Note that although devices are not accessed in intervals less than a specified cycle, they can be accessed in intervals greater than a specified cycle. (Only IN interrupt transfers are supported in USB1.0, but USB1.1 supports both IN and OUT interrupt transfers.)

Interrupt transfers can be used with both full-speed/low-speed devices.

The maximum data packet size that can be specified is 64 bytes for full-speed devices and 8 bytes for low-speed devices.

Each time a data receive action is repeated, the PID for the data packet toggles DATA0→DATA1→DATA0...

In an interrupt-in transfer, if the Host Controller generates an IN token and the device has data to transmit, the device sends a data packet, as illustrated in figure 2.24 (a) (left). If the device has no transmit data when an IN token is generated, the device issues NAK instead of sending a data packet, as shown in figure 2.24 (a) (right)

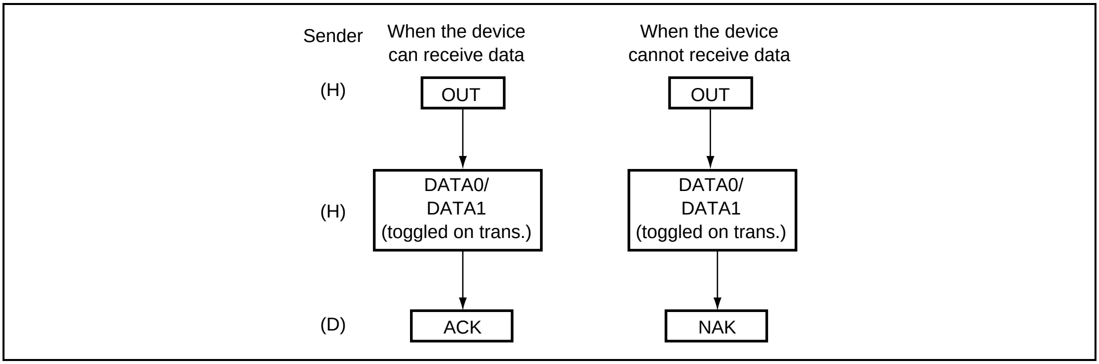

Sender
When the device can receive data
When the device cannot receive data
(H)
OUT
OUT
(H)
DATA0/ DATA1 (toggled on trans.)
DATA0/ DATA1 (toggled on trans.)
(D)
ACK
NAK

Figure 2.24 (a) Interrupt-In Transfer

In an interrupt-out transfer, the Host Controller sends an OUT token then data to the device. When the device has received the data, it sends an ACK packet, as illustrated in figure 2.24 (b) (left). If the device failed to receive data following the OUT token sent from the host controller, the device sends a NAK packet instead of an ACK packet, as shown in figure 2.24 (b) (right)

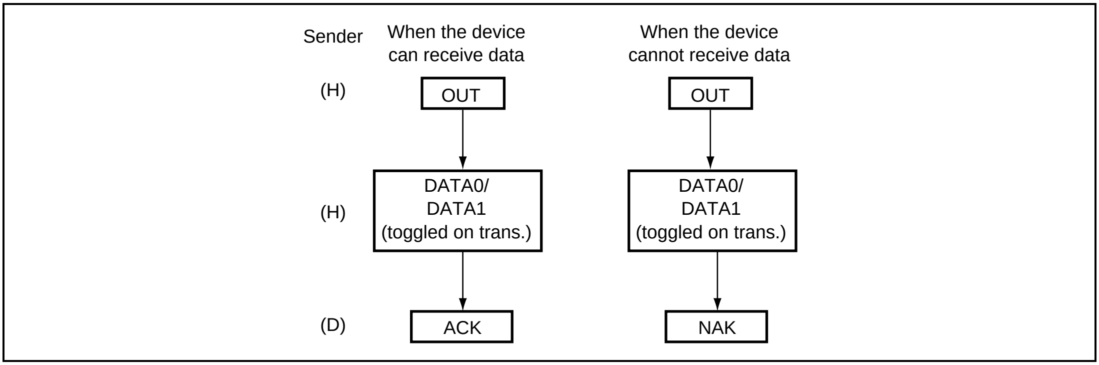

Sender
When the device can receive data
When the device cannot receive data
(H)
OUT
OUT
(H)
DATA0/ DATA1 (toggled on trans.)
DATA0/ DATA1 (toggled on trans.)
(D)
ACK
NAK

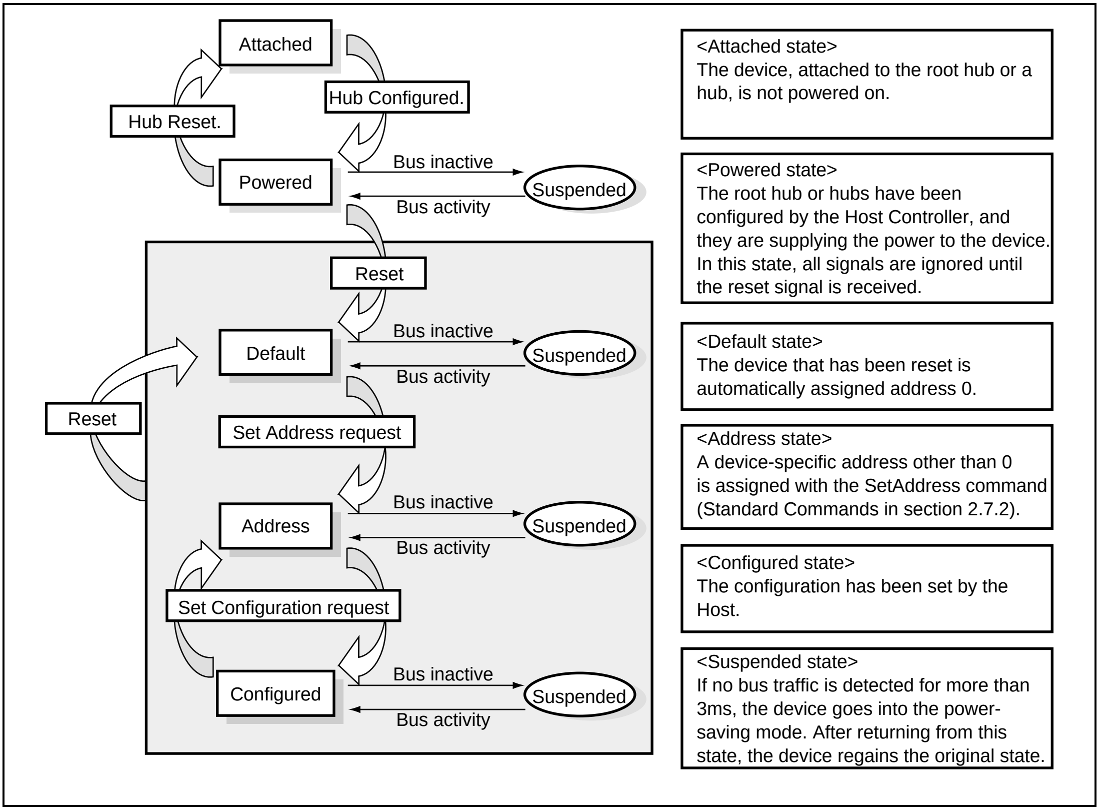
Figure 2.24 (b) Interrupt-Out Transfer

2.7 USB Device Framework

For plug-and-play, for the USB, detailed procedures are established from connecting the USB cable to configuring the system. This section explains those procedures.

2.7.1 Device States

USB devices can have the various states shown in figure 2.25. A device can be used only when it has transited to the configuration state.



<Attached state>
The device, attached to the root hub or a hub, is not powered on.

<Powered state>
The root hub or hubs have been configured by the Host Controller, and they are supplying the power to the device. In this state, all signals are ignored until the reset signal is received.

<Default state>
The device that has been reset is automatically assigned address 0.

<Address state>
A device-specific address other than 0 is assigned with the SetAddress command (Standard Commands in section 2.7.2).

<Configured state>
The configuration has been set by the Host.

<Suspended state>
If no bus traffic is detected for more than 3ms, the device goes into the power-saving mode. After returning from this state, the device regains the original state.

Figure 2.25 USB Device State

2.7.2 Device Request

For a device to be able to transit to the configuration state, it must respond to the commands issued by the Host Controller. Commands issued by the Host Controller are called device requests, and their format is defined by the USB standard. The Host Controller issues device requests in the setup stage in a control transfer.

Three types of device requests are available:

Standard commands

These commands are defined in the USB standard. All devices must support these commands. Table 2.5 shows a list of standard commands.

For details on standard commands, refer to the standards documentation.

Table 2.5 List of Standard Commands

Command Name	Function	Data Stage	Direction of Data Stage
Clear_Feature (Endpoint_stall)	Clears the endpoint stall.	No	
Clear_Feature (Device_Remote_Wakeup)	Clears the device remote wakeup feature.	No	
Get_Configuration	Gets configuration information.	Yes	IN
Get_Descriptor (Device)	Gets device descriptor information.	Yes	IN
Get_Descriptor (Config)	Gets configuration descriptor information.	Yes	IN
Get_Descriptor (String)	Gets string descriptor information.	Yes	IN
Get_Interface	Gets interface information.	Yes	IN
Get_Status(Device)	Gets device status information.	Yes	IN
Get_Status(Interface)	Gets interface status information.	Yes	IN
Get_Status(EndPoint)	Gets endpoint status information.	Yes	IN
Set_Address	Sets the device address.	No	

Command Name	Function	Data Stage	Direction of Data Stage
Set_Descriptor (Device)	Sets the device descriptor.	Yes	Out
Set_Descriptor (Config)	Sets the configuration descriptor.	Yes	Out
Set_Descriptor (String)	Sets the string descriptor.	Yes	Out
Set_Configuration	Sets configuration.	No	
Set_Feature (EndPoint_Stall)	Sets the endpoint to the Stall stage.	No	
Set_Feature (Device_Remote_Wakeup)	Sets the device to the wakeup state.	No	
Set_Interface	Sets an interface.	No	
Sync_Frame	Posts a specific frame number on the endpoint during an Isochronous transfer (if a special number is required).	Yes	Out

Class command

Class commands other than hub commands are established by corporate groups, subject to certification by the USB-IF (USB Implementers Forum). Several classes exist: audio class, common class, HID (Human Interface Device) class, and printer class.

Vendor command

Vendor commands can be defined freely by device designers, provided that the commands conform to the same format as other commands.

2.8 Descriptor

Each USB device is associated with what is called descriptor information that indicates the type, characteristics, and attributes of the device itself. By obtaining device information on a device, the Host Controller can recognize the type of device that is connected to a given bus.

Standard USB devices have the following descriptors: device, configuration, interface, and endpoint.

These descriptors are described in tables 2.6, 2.7, 2.8, and 2.9.

Table 2.6 Device Descriptor

Field	Size (in bytes)	Description
bLength	1	Descriptor size (fixed at 0x12)
bDescriptorType	1	Descriptor type (fixed at 0x01)
bcdUSB	2	USB version, represented in BCD
bDeviceClass	1	Class code: 0: no class; 0xFF: vendor class 1 to 0xFE: special class
bDeviceSubClass	1	Subclass code
bDeviceProtocol	1	Protocol code: 0: no specific protocol used 0xFF: vendor-specific protocol
bMaxPacketSize0	1	Maximum packet for endpoint 0
idVendor	2	Vendor ID (assigned to manufacturers by the USB-IF)
idProduct	2	Product ID (assigned to each device by manufacturer)
bcdDevice	2	Device version, represented in BCD
iManufacturer	1	Index to a string descriptor indicating the manufacturer’s name
iProduct	1	Index to a string descriptor indicating the device name
iSerialNumber	1	Index to a string descriptor indicating the serial number of the device
bNumConfigurations	1	Number of configurable devices

Note: USB Implementers Forum

Table 2.7 Configuration Descriptor

Field	Size (in bytes)	Description
bLength	1	Descriptor size (fixed at 0x09)
bDescriptorType	1	Descriptor type (fixed at 0x02)
wTotalLength	2	Total length of descriptor
bNumInterface	1	Number of interfaces associated with descriptor
bConfiguration Value	1	Argument value (1 or higher) for the selection of this descriptor using Set_Configuration
iConfiguration	1	Index to a string descriptor
bmAttributes	1	Device power supply Bit 7: bus power; bit 6: self-power; bit 5: remote wakeup; bits 4 to 0: reserved
MaxPower	1	Specifies the maximum bus power consumption in units of 2 mA.

Table 2.8 Interface Descriptor

Field	Size (in bytes)	Description
bLength	1	Descriptor size (fixed at 0x09)
bDescriptorType	1	Descriptor type (fixed at 0x04)
bInterfaceNumber	1	Zero-base index number that represents this interface in the configuration□
bAlternateSetting	1	An argument value for the selection of alternate settings using Set_Interface.
bNumEndpoints	1	Number of endpoints associated with a device (exclusive of endpoint 0)
bInterfaceClass	1	Class code 0: no class; 0xFF: vendor class; 1 to 0xFE: special class
bInterfaceSubClass	1	Subclass code
bInterfaceProtocol	1	Protocol code : no specific protocols used 0xFF: vendor-specific protocol
iInterface	1	Index to the string descriptor representing this interface

Table 2.9 Endpoint Descriptor

Field	Size (in bytes)	Description
bLength	1	Descriptor size (fixed at 0x07)
bDescriptorType	1	Descriptor type (fixed at 0x05)
bEndpointAddress	1	Endpoint address: bit 7: direction (0:OUT 1:IN); bits 6 to 4: reserved (0); bits 3 to 0: endpoint number
bmAttributes	1	Endpoint transfer method: bits 7 to 2: reserved (0); bits 1 to 0: transfer method (0: control, 1: Isochronous , 2: bulk, 3: interrupt)
wMaxPacketSize	2	Maximum packet size
bInterval	1	Specifies polling intervals in units of ms. Specify 1 for Isochronous transfers. Ignored for bulk or control transfers.

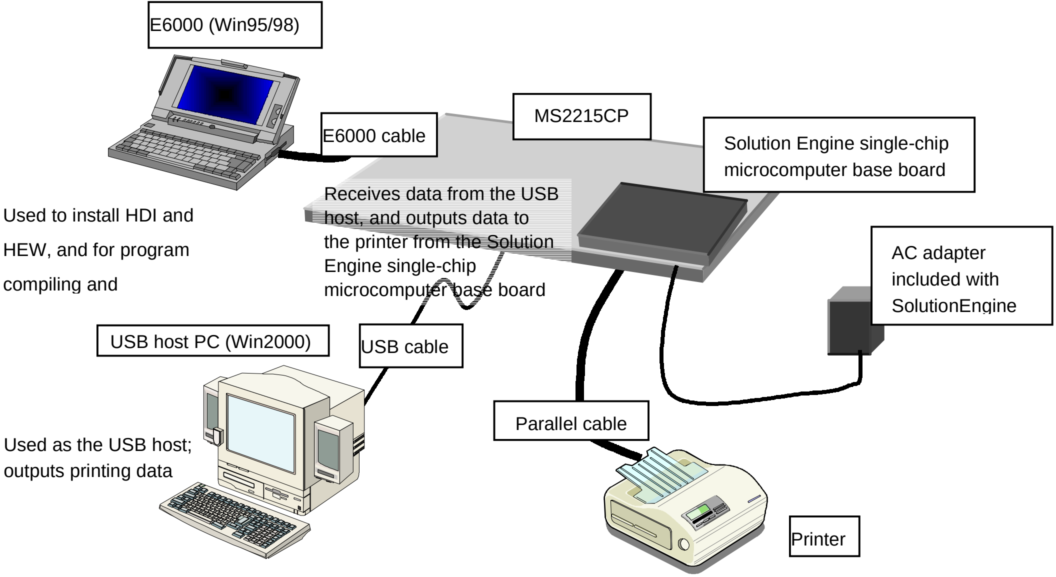
Section 3 Development Environment

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

- H8S/2215 Solution Engine (hereafter called the MS2215CP; type number: MS2215CP01_C/S) manufactured by Hitachi ULSI Systems Co., Ltd.
- Solution Engine Single-Chip Microcomputer Base Board (hereafter called the base board; type number: MSSCBB01) manufactured by Hitachi ULSI Systems Co., Ltd.
- E6000 (type number: HS2214EPI61H) Emulator manufactured by Hitachi, Ltd.
- H8S/2215 Series TFP120 User System Interface Cable (hereafter called the H8S/2215 user cable; type number: HS2215ECN61H) manufactured by Hitachi, Ltd.
- PC (Windows 95/98) equipped with an ISA, PCI, or PCMCIA slot
- PC (Windows 2000/Windows Millennium Edition or Mac OS9) to serve as the USB host
- Parallel-port printer
- USB cable
- Parallel cable
- Hitachi Debugging Interface (hereafter called the HDI) manufactured by Hitachi, Ltd.
- Hitachi Embedded Workshop (hereafter called the HEW) manufactured by Hitachi, Ltd.

3.1 Hardware Environment

Figure 3.1 shows device connections.



E6000 (Win95/98)
E6000 cable
MS2215CP
Solution Engine single-chip microcomputer base board
Receives data from the USB host, and outputs data to the printer from the Solution Engine single-chip microcomputer base board
Used to install HDI and HEW, and for program compiling and
AC adapter included with SolutionEngine
USB host PC (Win2000)
USB cable
Parallel cable
Used as the USB host; outputs printing data
Printer

Figure 3.1 Device Connections

1. MS2215CP

   Some jumper settings on the MS2215CP board must be changed from those at shipment. Before turning on the power, ensure that the jumpers are set as follows. There is no need to change any other jumpers.

Table 3.1 Jumper Settings

At Shipment	After Change	Jumper Function
J9 1-2: Closed	J9 2-3: Closed	Switches the EXTAL48 pin level
J14 1-2: Closed	J14 1-2: Open	Enables SRAM
J15 1-2: Closed	J15 1-2: Open	Enables LED

2. Solution Engine single-chip microcomputer base board

   For an explanation of connection with the MS2215CP, please refer to the instruction manual for the base board. This base board is not included with the SolutionEngine, and must be purchased separately. The base board has a 26-pin Centronics interface connector (CN4). If the parallel cable has a different type of connector, create a conversion connector according to table 6.7, Connector Signal Assignment, in section 6.5.2, Centronics Interface of the instruction manual for the base board.

3. USB host PC

   A PC with Windows 2000 installed and with a USB port is used as the USB host. This system uses printer-class device drivers installed as a standard part of the Windows 2000 system, and so there is no need to install new drivers.

4. E6000

   The ISA is used for the communication interface between the E6000 PC and the E6000 emulator.

   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.

3.2 Software Environment

A sample program, as well as the compiler and linker used, are explained.

3.2.1 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 indicated in figure 3.2 below.

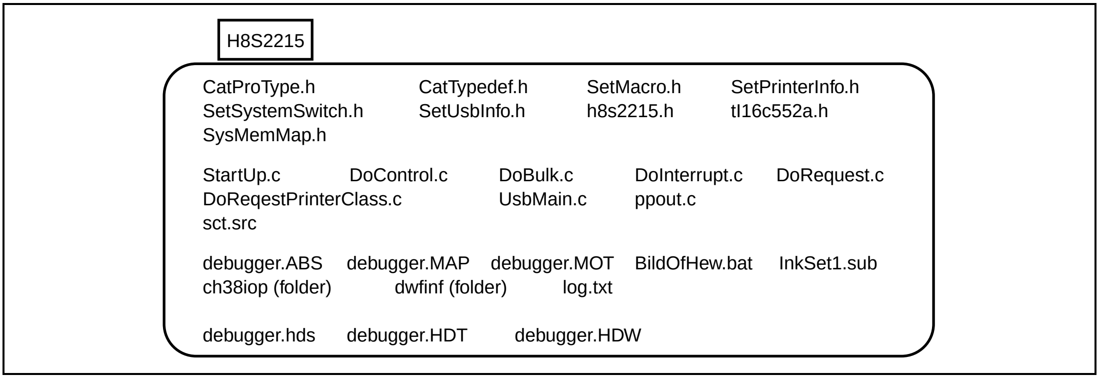


Figure 3.2 Files Included in the Folder

3.2.2 Compiling and Linking

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

Hitachi Embedded Workshop Version 1.0 (release 9) (hereafter 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 3.3)

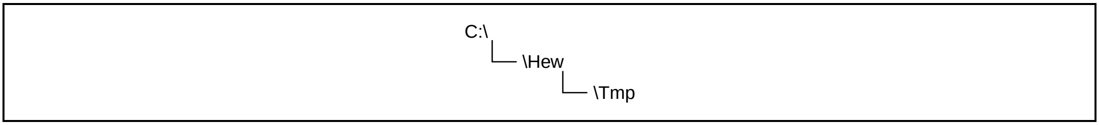


Figure 3.3 Creating a Working Folder

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 should be modified to the path to the copied folder). In addition to the sample program, this folder contains a batch file named BildOfHew.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 BildOfHew.bat is executed, compiling and linking are performed. As a result, a Motorola S-type format file named debugger.MOT is created within the folder. This is the executable file. 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 BildOfHew.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, the setting for the environment variable ch38 used by the compiler should be set to the folder of machine.h, and the setting of ch38tmp should specify the work folder for the compiler. The library setting should specify the path of c8s26a.lib.

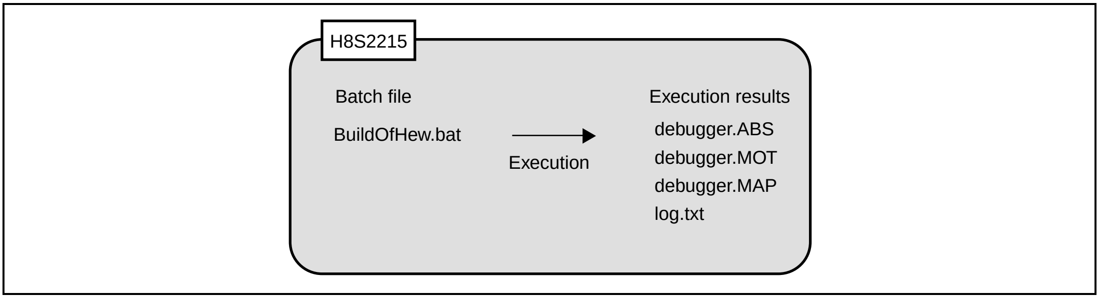
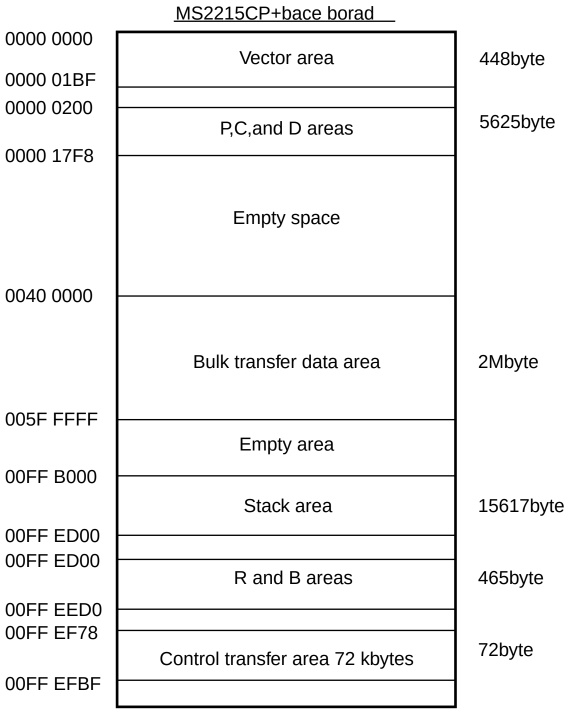


Figure 3.4 Compile Results

3.3 Loading and Executing the Program

Figure 3.5 shows the memory map for the sample program.



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

Figure 3.5 Memory Map

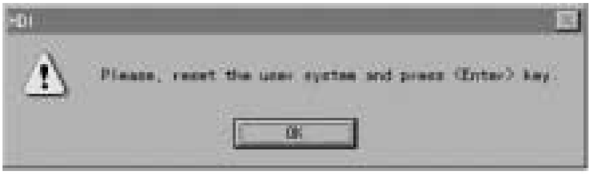
As shown in figure 3.5, this sample program allocates areas for vectors, P, C, and D to the on-chip ROM area (E6000 emulation memory) in area 1, the stack, B, R, and control transfer areas to the on-chip RAM, and the print data area to the SRAM. These memory allocations are specified by the InkSet1.sub file in the H8S2215 folder. When modifying the program allocation, this file must be modified.

3.3.1 Loading the Program

In order to load the sample program into the MS2215CP, 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.
- Turn on the power to the E6000 PC, to start up the machine.
- Execute debugger.hds in the H8S2215 folder.

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



Please, reset the user system and press (Enter) key.

Figure 3.6 Reset Request Dialog

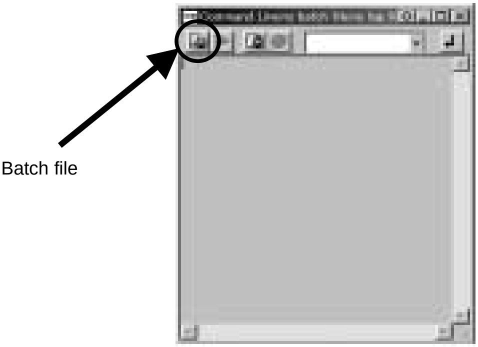


Figure 3.7 Command Line Input

3.3.2 Executing the Program

In order to execute the program which was loaded in section 3.3.1, Loading the Program, above, the program counter (PC) must be set appropriately.

Select Register Window from the View menu to open the Registers window. On double-clicking the numerical area of the register (PC) in the window, a dialog box appears, and the register value can be changed. Use this dialog box to set the PC to H'0000 0200.

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

3.4 Printing Procedure

With the program executed, insert the USB cable series B connected into the MS2215CP, and connect the series A connected at the opposite end to the USB host PC. After control transfer is completed, USB printing support is displayed below USB host controller in the device manager, and the host PC recognizes the MS2215CP as a printer device.

Next, the printer driver[*1] is installed. Open the printer from the Start menu Settings item, and double-click on the Add a printer icon. A setup wizard is started; in port selection, check USB001 Virtual Printer Port for USB[*2]. Specify the printer to be used (the manufacturer name and printer model). When the wizard processing is completed, a test print should be performed; if the driver is correctly installed, the printer will output a print test.

Notes: *1 In this sample program, bidirectional communication with the printer is not supported; please be sure to use a printer driver included as standard with Windows 2000.

*2 If a printer-class device has previously been connected to the host PC, the number may be different (USB002, USB003, etc.). In this case, select the highest-numbered port.

Section 4 Overview of the Sample Program

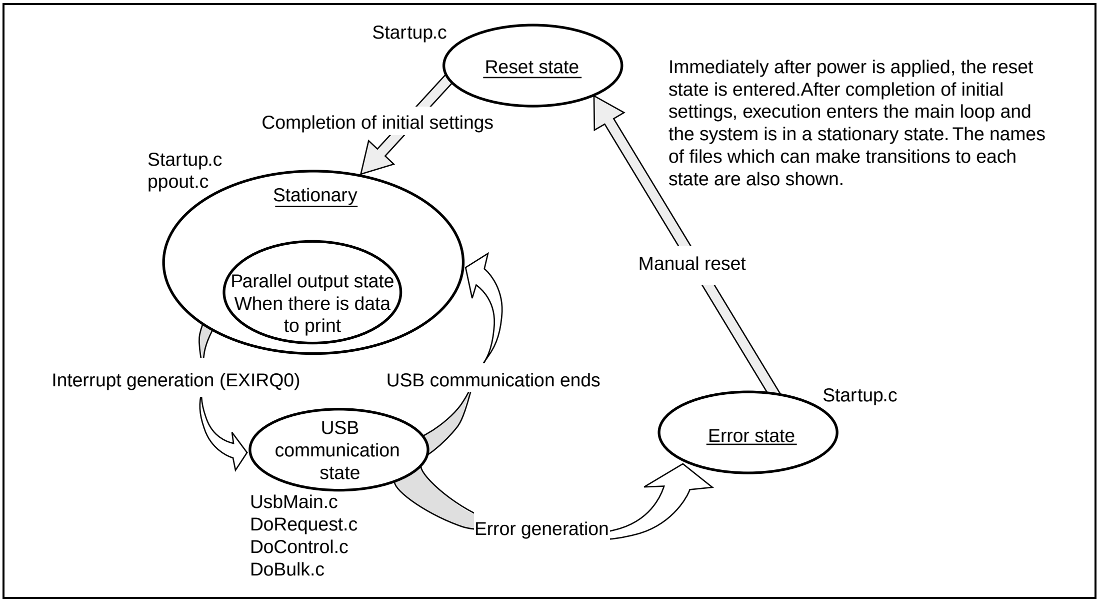
In this section, features of the sample program and its structure are explained. This sample program runs on the MS2215CP + base board, and initiates USB transfers by means of interrupts from the USB function module. Of the interrupts from modules in the H8S/2215, there are three interrupts related to the USB function module: EXIRQ0, EXIRQ1, and IRQ6, but in this sample program, only EXIRQ0 is used.

Features of this program are as follows.

- Control transfer can be performed.
- Bulk-out transfer can be used to receive data from the host controller.
- Bulk-in transfer can be used to send data to the host controller.
- The Ultra I/O mounted on the MS2215CP can be used to output data to a printer.

4.1 State Transition Diagram

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



Immediately after power is applied, the reset state is entered.After completion of initial settings, execution enters the main loop and the system is in a stationary state. The names of files which can make transitions to each state are also shown.

Figure 4.1 State Transition Diagram

- Reset State

  Upon power-on reset and manual reset, this state is entered. In the 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. Here, the presence of printing data from the host is constantly monitored; if there is data, the parallel output state is entered, and data is output to the printer.

- USB Communication State

  In the stationary state, when an interrupt from the USB module occurs, this state is entered. In the USB communication state, data transfer is performed by a transfer method according to the type of interrupt. The interrupts used in this sample program are indicated by interrupt flag register 0 (UIFR0) to interrupt flag register 3 (UIFR3), and there are nine interrupt types in all. When an interrupt factor occurs, the corresponding bits in UIFR0 to UIFR3 are set.

- Error State

  When an error occurs while in the USB communication state, this state is entered. In the case of a transition to the error state, there is a problem with the USB communication contents. When communication is performed normally, there are no transitions to the error state. If the error state is entered, the firmware should be reexamined. In order to recover from the error state, perform a power-on reset or a manual reset.

4.2 USB Communication State

The USB communication state can be further divided into three states according to the transfer type (see figure 4.2). When an interrupt occurs, first there is a transition to the USB communication state, and then there is further branching to a transfer state according to the interrupt type. The branching method is explained in section 5, Sample Program Operation.

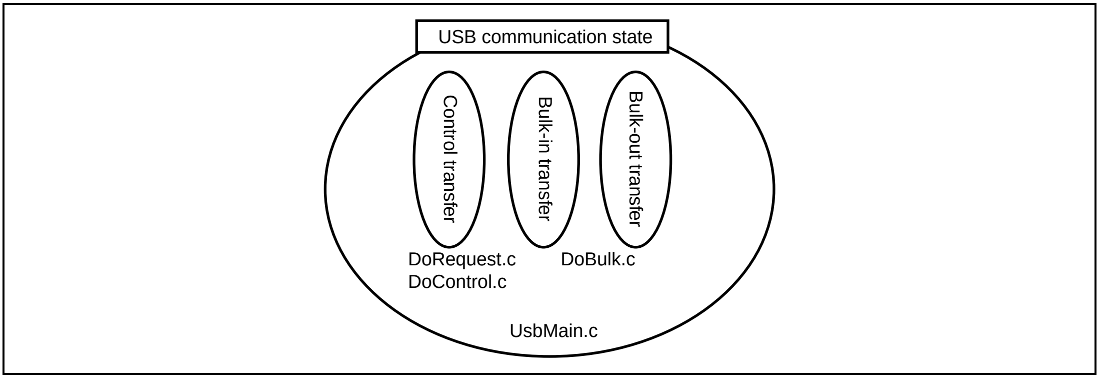


Figure 4.2 USB Communication State

4.3 File Structure

This sample program consists of seven source files and nine header files. The overall file structure is shown in table 4.1. Each function is arranged in one file by transfer method or function type.

Table 4.1 File Structure

Filename	Main purpose
StartUp.c	Makes microcomputer initial settings Clears ring buffer
UsbMain.c	Discriminates interrupt factors Sends/receives packets
DoRequest.c	Processes setup commands issued by host
DoControl.c	Executes control transfer
DoBulk.c	Executes bulk transfer
DoRequestPrinter Class.c	Processes printer-class commands
ppout.c	Controls ring buffer Initializes printer Outputs data to printer
CatProType.h	Declares prototypes
CatTypedef.h	Defines basic structures used in the USB firmware
SysMemMap.h	Defines MS2215CP memory map addresses
h8s2215.h	Defines H8S/2215 registers
tl16c552a.h	Defines the TL16C552A registers
SetSystemSwitch.h	Sets system operation
USBFunctionModu.h	Defines the USB function module registers
SetMacro.h	Defines macros
SetUsbInfo.h	Makes initial settings of variables needed to support USB
SetPrinterInfo.h	Makes initial settings of variables needed to support bulk-only transport

4.4 Purposes of Functions

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

Table 4.2-1 UsbMain.c

File in Which Stored	Function Name	Purpose
UsbMain.c	BranchOfInt	Discriminates interrupt factors, and calls function according to interrupt
	GetPacket	Writes data transferred from the host controller to RAM.
	GetPacket4	Writes data transferred from the host controller to RAM in longwords. Ring buffer support version.
	GetPacket4S	Writes data transferred from the host controller to RAM in longwords. High-speed version. (not used by this sample program)
	PutPacket	Writes data for transfer to the host controller to the USB module
	PutPacket4	Writes data for transfer to the host controller to the USB module in longwords. Ring buffer support version.
	PutPacket4S	Writes data for transfer to the host controller to the USB module in longwords. High-speed version. (not used by this sample program)
	SetControlOutContents	Overwrites data with that sent from the host
	SetUsbModule	Makes USB module initial settings
	ActBusReset	Clear FIFO on receiving bus reset
	ActBusVcc	Pulls up D+ and controls USB module when the USB cable is connected or disconnected
	ConvRealn	Reads data of a specified byte length from a specified address
	ConvReflexn	Reads data of a specified byte length from specified addresses, in reverse order

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

Table 4.2-2 StartUp.c

File in Which Stored	Function Name	Purpose
StartUp.c	SetPowerOnSection	Sets BSC, terminals, and interrupt controller, calls initialization routines, and shifts to the main loop
	_INITSCT	Copies variables with initial values to RAM work area
	InitMemory	Clears RAM area used in bulk communication
	InitSystem	Specifies the USB clock, system interrupts, and masks

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

Table 4.2-3 ppout.c

File in Which Stored	Function Name	Purpose
	ActPrintOut	Monitors the empty space in the buffer and stops bulk-out transfer if necessary Calls bulk-out functions
ppout.c	LptMain	Monitors the empty space in the buffer and restarts bulk-out transfer if necessary Passes the read pointer as argument to LptPortWrite
	LptPortOpen	Initializes printer
	LptPortWrite	Outputs data from parallel port

In ppout.c, print data stored in RAM is written to the TL16C552A register, and strobe and other signals are controlled to output data to the printer.

Table 4.2-4 DoRequest.c

File in Which Stored	Function Name	Purpose
DoRequest.c	DecStandardCommands	Decodes command issued by host controller, processes standard commands
	DecVenderCommands	Processes vendor 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. Because vendor commands are not used, DecVenderCommands does not perform any action. In order to use a vendor command, the customer should develop a program.

Table 4.2-5 DoControl.c

File in Which Stored	Function Name	Purpose
DoControl.c	ActControl	Performs setup stage for control transfer
	ActControlIn	Performs data stage, status stage for control transfer (data stage transferred in in direction)
	ActControlOut	Performs data stage, status stage for control transfer (data stage transferred in out direction)

When a control transfer interrupt (EPOoTS) is input, ActControl acquires the command, and decoding is performed by DecStandardCommands. Next, the data stage and status stage are performed by ActControlIn and ActControlOut, according to the command type.

Table 4.2-6 DoBulk.c

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

Processing related to bulk transfer is performed. ActBulkInReady is used only in bulk-in transfer.

Table 4.2-7 DoRequestPrinterClass.c

File in Which Stored	Function Name	Purpose
DoRequestPrinterClass.c	DecPrinterClassCommands	Processes printer-class command

Processing for printer class commands is performed. In this sample program, an IEEE 1284 database ID is not used, and so 0 is output. When using an IEEE 1284 device ID, the output value should be set by the customer.

Figure 4.3 shows the interrelationship between the functions explained in table 4.2. The upper-side functions can call the lower-side functions. Also, multiple functions can call the same function. In the stationary state, SetPowerOnSection calls other functions, and in the case of a transition to the USB communication state which occurs on an interrupt, BranchOfInt calls other functions. Figure 4.3 shows the hierarchical relation of functions; there is no order for function calling. For information on the order in which functions are called, please refer to the flow charts of section 5, Sample Program Operation.

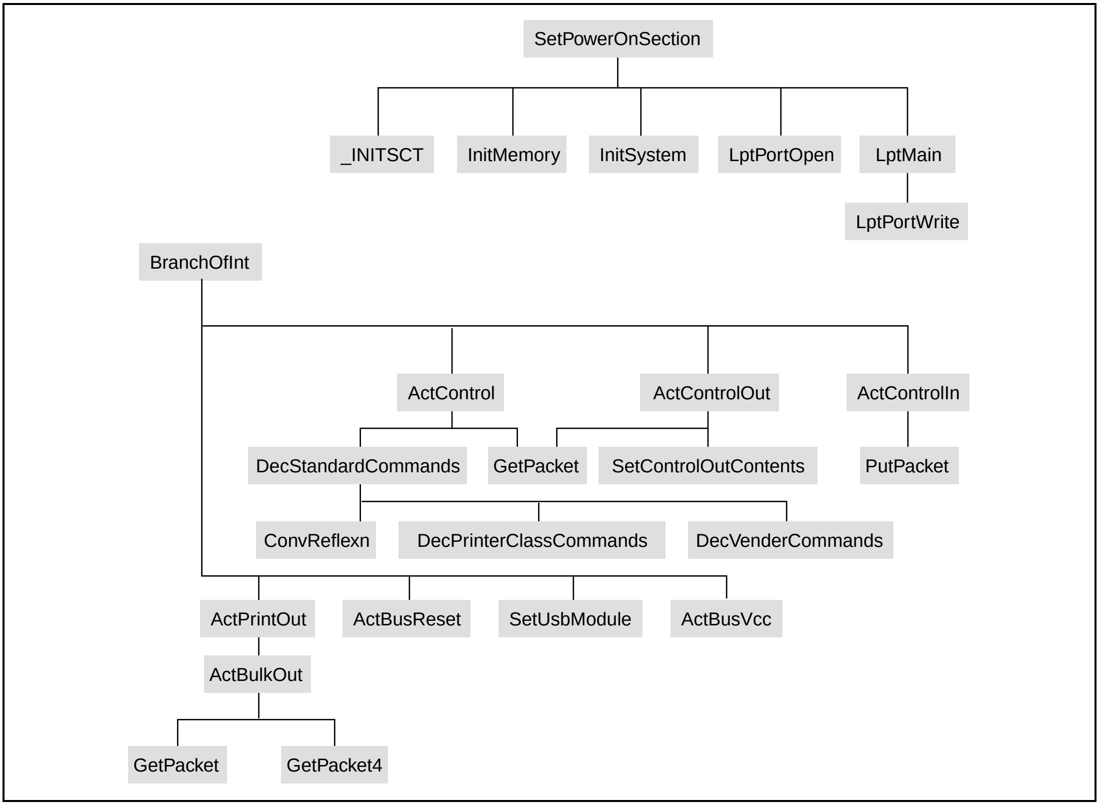


Figure 4.3 Interrelationship between Functions

Section 5 Sample Program Operation

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

5.1 Main Loop

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

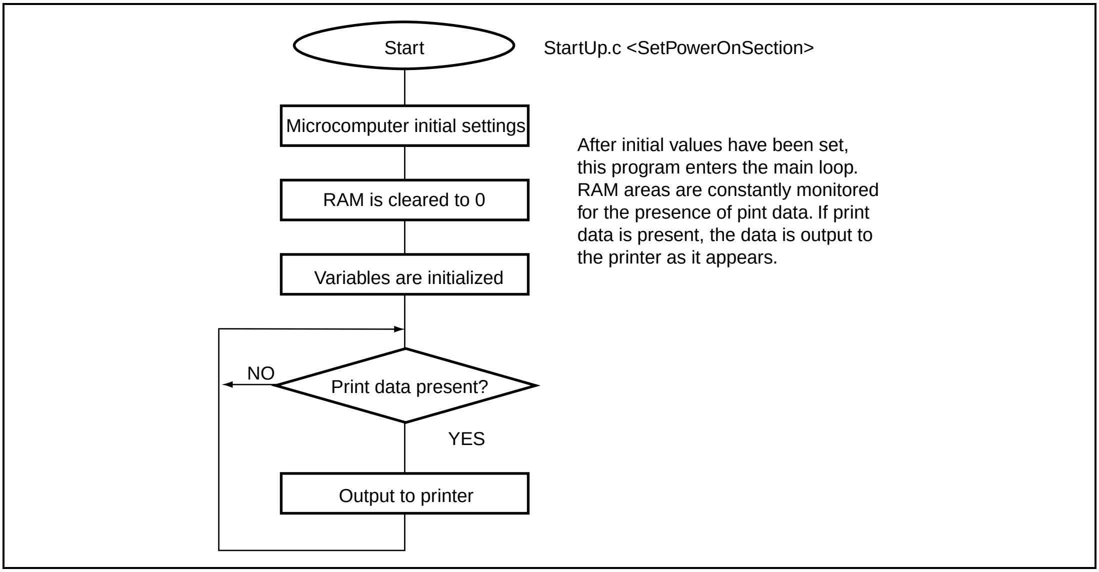

Start
StartUp.c <SetPowerOnSection>
Microcomputer initial settings
RAM is cleared to 0
Variables are initialized
NO
Print data present?
YES
Output to printer
After initial values have been set,
this program enters the main loop.
RAM areas are constantly monitored
for the presence of pint data. If print
data is present, the data is output to
the printer as it appears.

Figure 5.1 Main Loop

5.2 Types of Interrupts

As explained in section 5.1, State Transition Diagram, the interrupts used in this sample program are indicated by the interrupt flag registers 0 to 3 (UIFR0 to UIFR3); there are a total of nine types of interrupts. When an interrupt factor occurs, the corresponding bits in the interrupt flag register are set to 1, and an EXIRQ0 interrupt request is sent to the CPU. In the sample program, the interrupt flag registers are read as a result of this interrupt request, and the corresponding USB communication is performed. Figure 5.2 shows the interrupt flag registers and their relation to USB communication.

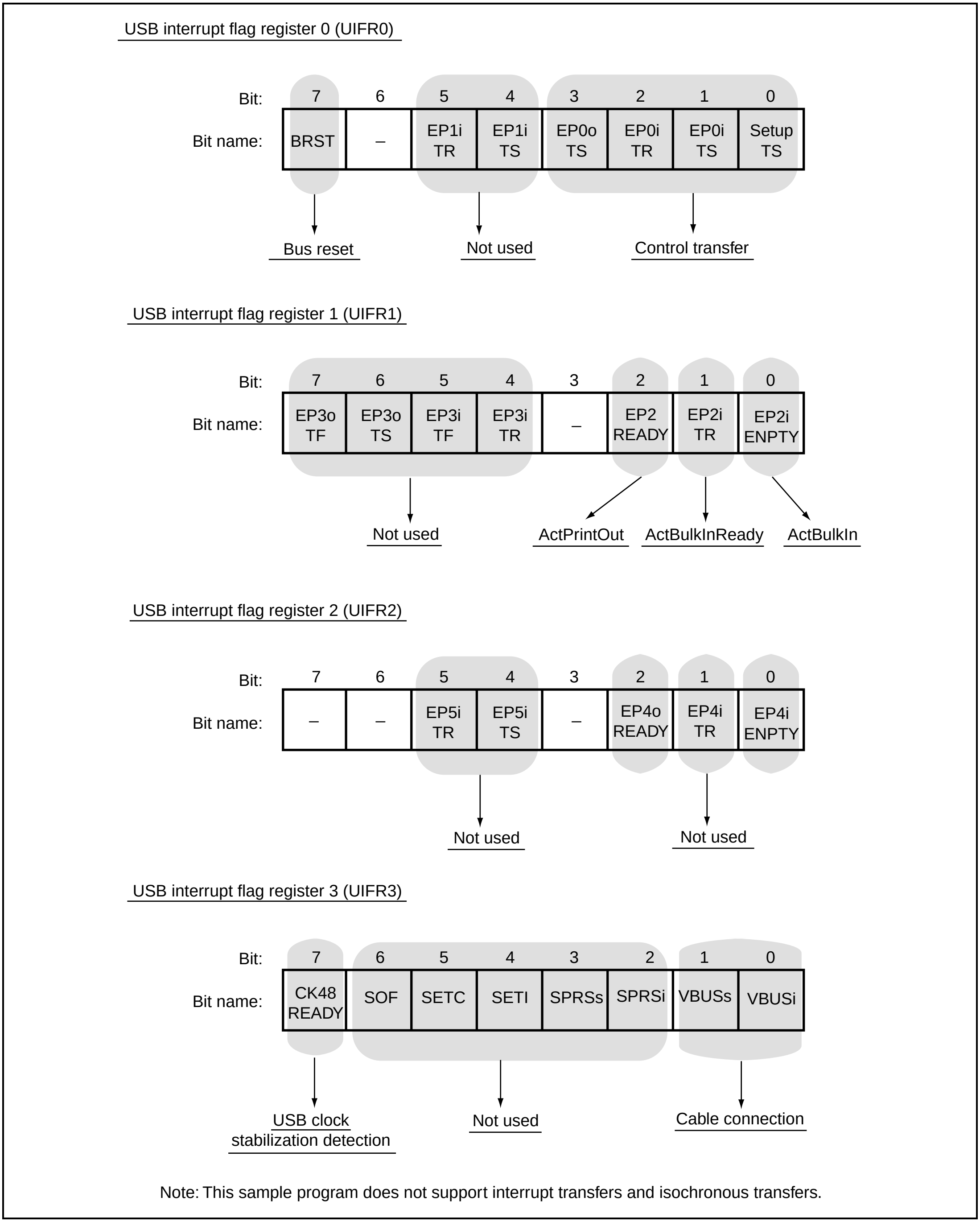


Figure 5.2 Types of Interrupt Flags

5.2.1 Method of Branching to Different Transfer Processes

In this sample program the transfer method is determined by the type of interrupt from the USB module as describe in section 4, Overview of the Sample Program. Branching to the different transfer methods is executed by BranchOfInt in UsbMain.c. Table 5.1 shows the relations between the types of interrupts and the functions called by BranchOfInt.

Table 5.1 Interrupt Types and Functions Called on Branching

Register Name	Bit	Bit Name	Name of Function Called
USB interrupt flag register 0 (UIFR0)	7	BRST	ActBusReset
	6	—	—
	5	EP1i TR	—
	4	EP1i TS	—
	3	EP0o TS	ActControlIn, ActControlOut
	2	EP0i TR	ActControlOut
	1	EP0i TS	ActControlIn, ActControlOut
	0	SETUP TS	ActControl
USB interrupt flag register 1 (UIFR1)	7	EP3o TF	—
	6	EP3o TS	—
	5	EP3i TF	—
	4	EP3i TR	—
	3	—	—
	2	EP2o Ready	ActPrintOut
	1	EP2i TR	ActBulkIn
	0	EP2i EMPTY	ActBulkInReady
USB interrupt flag register 3 (UIFR3)	7	CK48 Ready	ActBusReset
	6	SOF	SetUSBModule
	5	SETC	—
	4	SETI	—
	3	SPRSs	—
	2	SPRSi	—
	1	VBUSs	—
	0	VBUSi	ActBusVcc

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

In the H8S/2215 hardware manual, operation of the USB function module when an interrupt occurs, and a summary of operation on the application side, are described. From the next section, details of application-side firmware are explained for each USB transfer method.

5.3 USB Operating Clock Stabilization Interrupt

This interrupt occurs when the USB operating clock (48 MHz) stabilization time is automatically counted after USB module stop is canceled. After receiving the interrupt, the sample program writes the endpoint configuration information to the USB endpoint information registers (UEPIR00_0 to 22_4), makes necessary interrupt settings, and waits for USB cable connection.

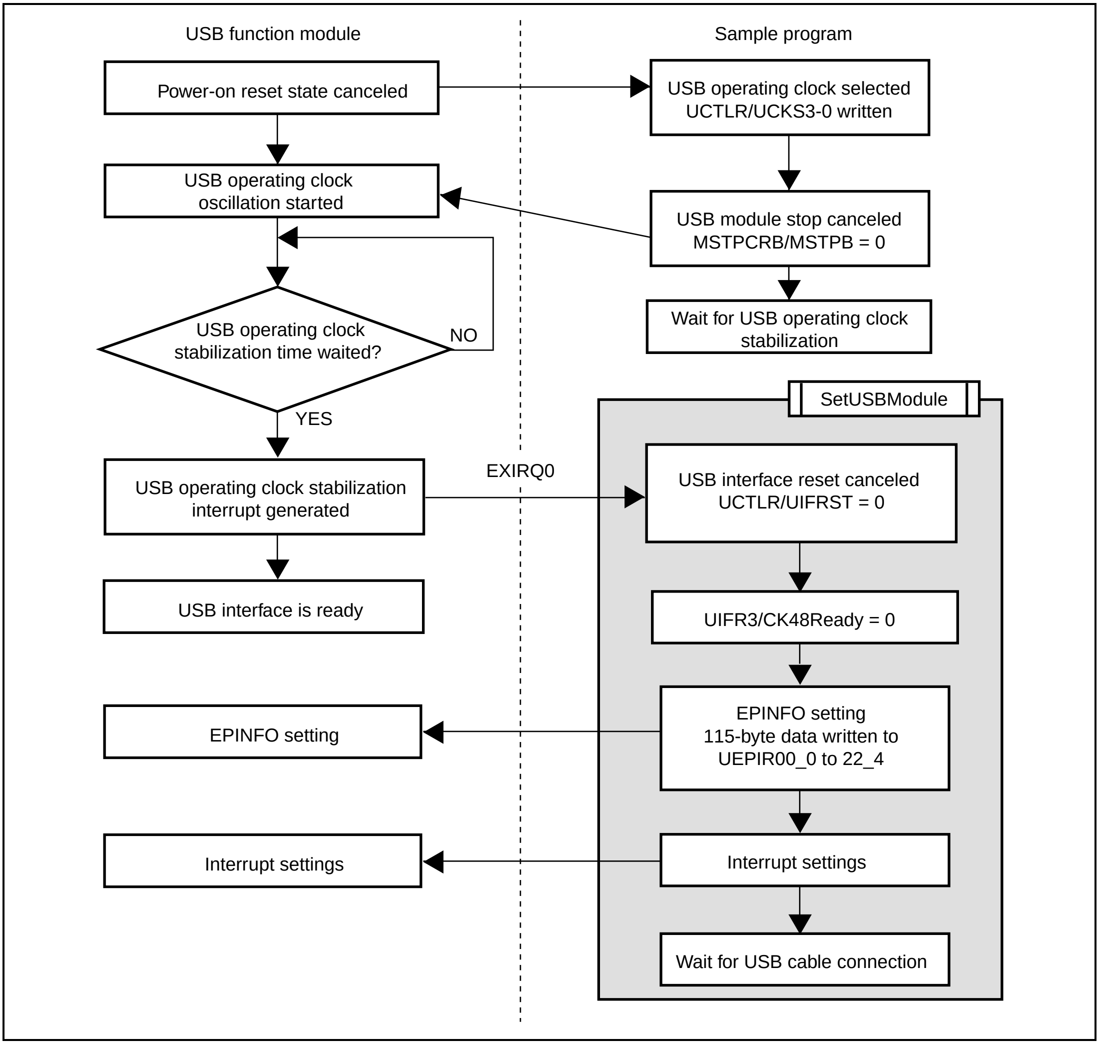


Figure 5.3 USB Operating Clock Stabilization Interrupt

5.3.1 Endpoint Configuration

In the USB function module in the H8S/2215, the endpoint configuration can be specified at initialization by software. The following transfer types can be specified:

- Control transfer: One endpoint
- Bulk-in transfer: Two endpoints
- Bulk-out transfer: Two endpoints
- Interrupt-in transfer: Two endpoints
- Isochronous-in transfer: One endpoint
- Isochronous-out transfer: One endpoint

The endpoint number, interface number, alternate number, and maximum packet size can be specified for the above transfers (excluding control transfer) with the USB endpoint information registers (UEPIRs).

Table 5.2 shows transfer types and their corresponding UEPIRs.

Table 5.2 Transfer Types and UEPIRs

Transfer Type	Endpoints	Corresponding UEPIRs
Control transfer	1	00
Interrupt-in transfer	2	01 and 02
Bulk-in transfer	2	02 and 20
Bulk-out transfer	2	03 and 21
Isochronous-in transfer	1	04, 06, 08, 10, 12, 14, 16, and 18
Isochronous-out transfer	1	05, 07, 09, 11, 13, 15, 17, and 19

The H8S/2215 Hardware Manual assumes that endpoint information is configured based on the Bluetooth standard. Figure 5.4 shows the comparison between the endpoint configuration used by this sample program and the endpoint numbers described in the H8S/2215 Hardware Manual.

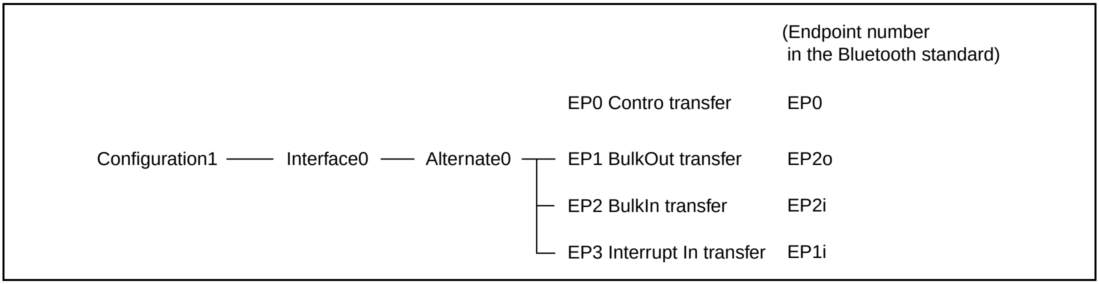


Figure 5.4 Endpoint Configuration in the Sample Program

Table 5.3 shows the UEPIR00_0 to 22_4 settings for the endpoint configuration shown in figure 5.4. Dummy data (0) must be written to the unused endpoints.

Table 5.3 UEPIR Settings

UEPIR	Set Value (Hexadecimal)	Transfer Type	EP No.	Interface No.	Alternate No.	Maximum Packet 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_15_40_00_02	BulkIn	2	0	0	64
03	14_10_40_00_03	BulkOut	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	BulkIn	0	0	0	0
21	04_10_00_00_15	BulkOut	0	0	0	0
22	04_10_00_00_16	Interrupt In	0	0	0	0

5.4 Interrupt on Cable Connection (VBUS)

This interrupt occurs when the cable of the USB function module is connected to the host controller. On the application side, after completion of initial microcomputer settings, a general-purpose output port is employed to pull-up the USB data bus D+. By means of this pull-up, the host controller recognizes that the device has been connected. (figure 5.5)

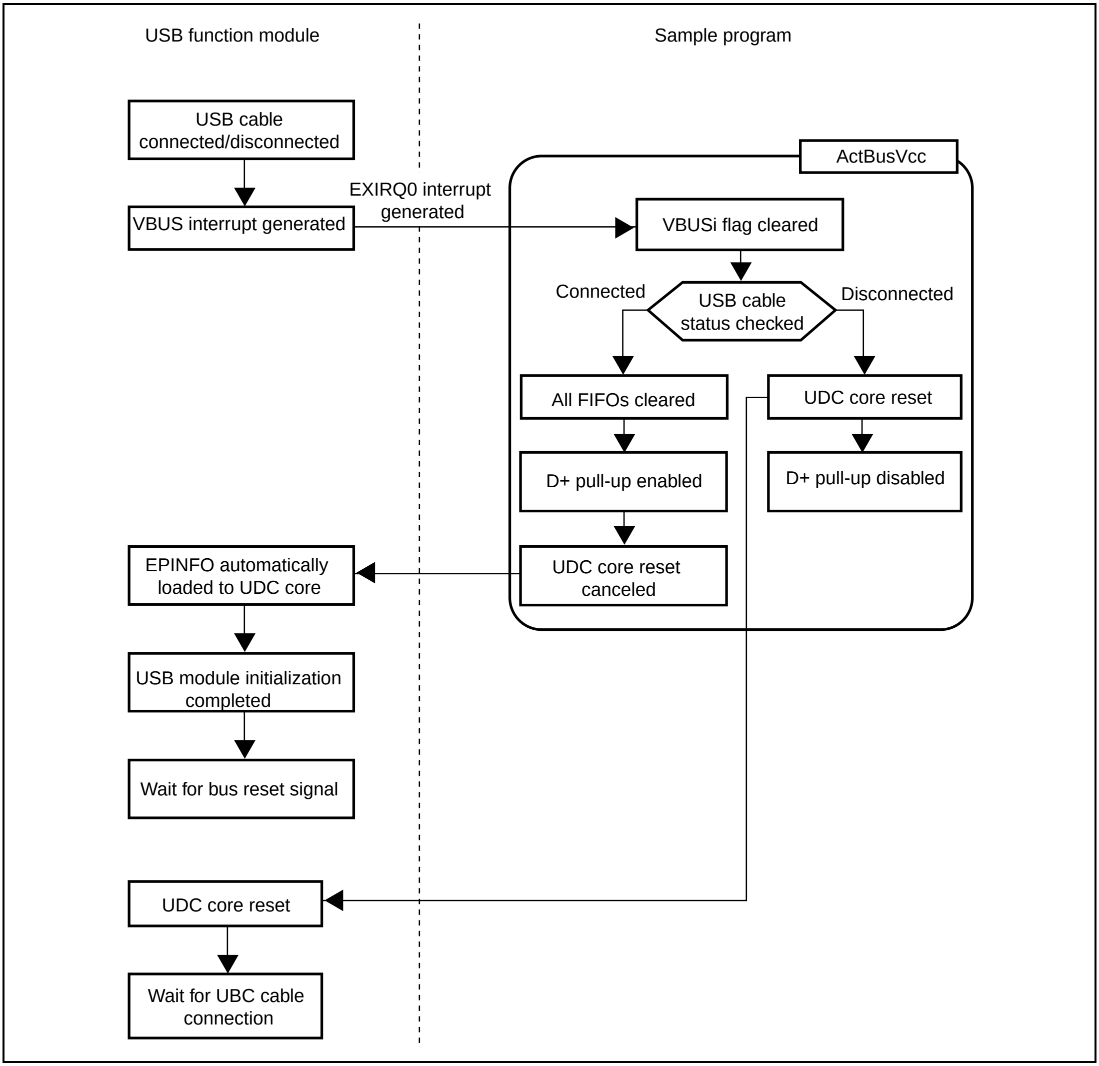


Figure 5.5 Interrupt on Cable Connection

5.5 Bus Reset Interrupt (BRST)

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

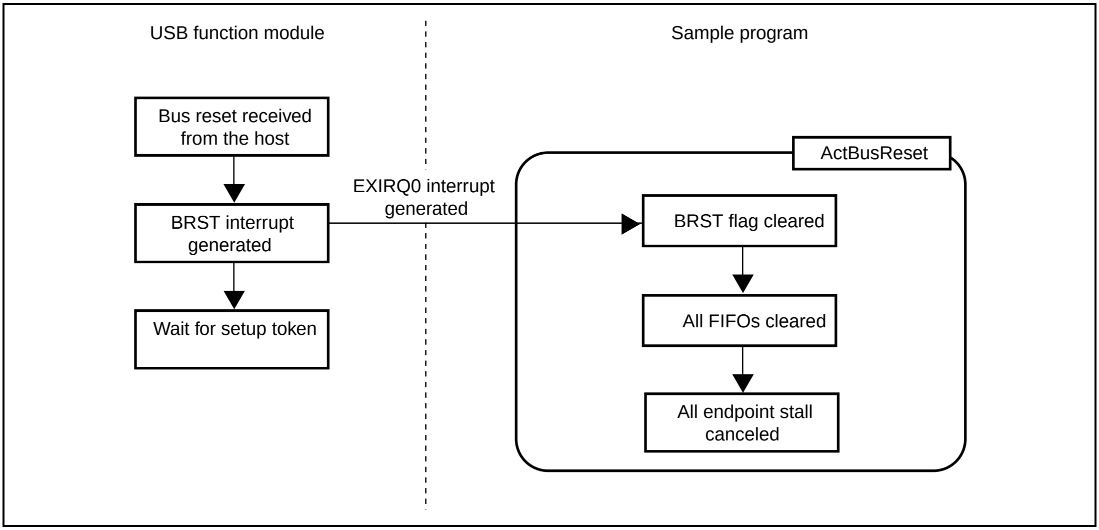


Figure 5.6 Bus Reset Interrupt

5.6 Control Transfers

In control transfers, bits 0 to 3 of the interrupt flag registers are used. Control transfers can be divided into two types according to the direction of data in the data stage. (figure 5.7) In the data stage, data transfers from the host controller to the USB function module are control-out transfers, and transfers in the opposite direction are control-in transfers.

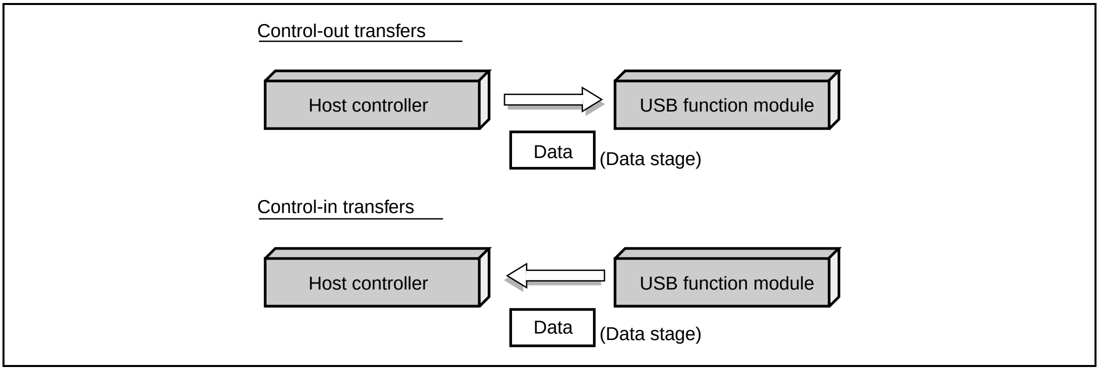


Figure 5.7 Control Transfers

Control transfers consist of three stages: setup, data (no data is possible), and status (figure 5.8). Further, the data stage consists of multiple bus transactions.

In control transfers, stage changes are recognized through the reversal of the data direction. Hence the same interrupt flag is used to call a function to perform control-in or control-out transfers (cf. Table 5.1). For this reason, the firmware must use states to manage the type of control transfer currently being performed, whether control-in or control-out, (figure 5.8) and must call the appropriate function. States in the data stage (TRANS_IN and TRANS_OUT) are determined by commands received in the setup stage.

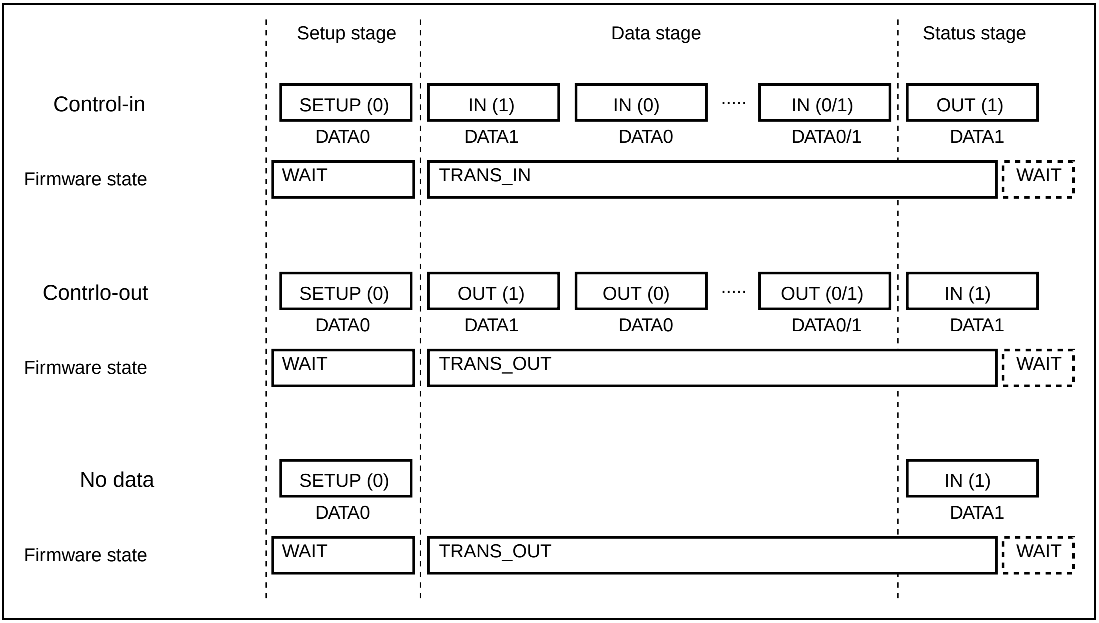


Figure 5.8 Status in Control Transfers

5.6.1 Setup Stage

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

• Commands for control-in transfers:	GetDescriptor (TRANS_IN)	Standard command
	GetDeviceID (TRANS_IN)	Class command
	GetPortStatus (TRANS_IN)	Class command
• Commands for control-out transfers:	SoftReset (TRANS_OUT)	Class command

Figure 5.9 shows operation of the sample program in the setup stage. The figure on the left shows operation of the USB function module.

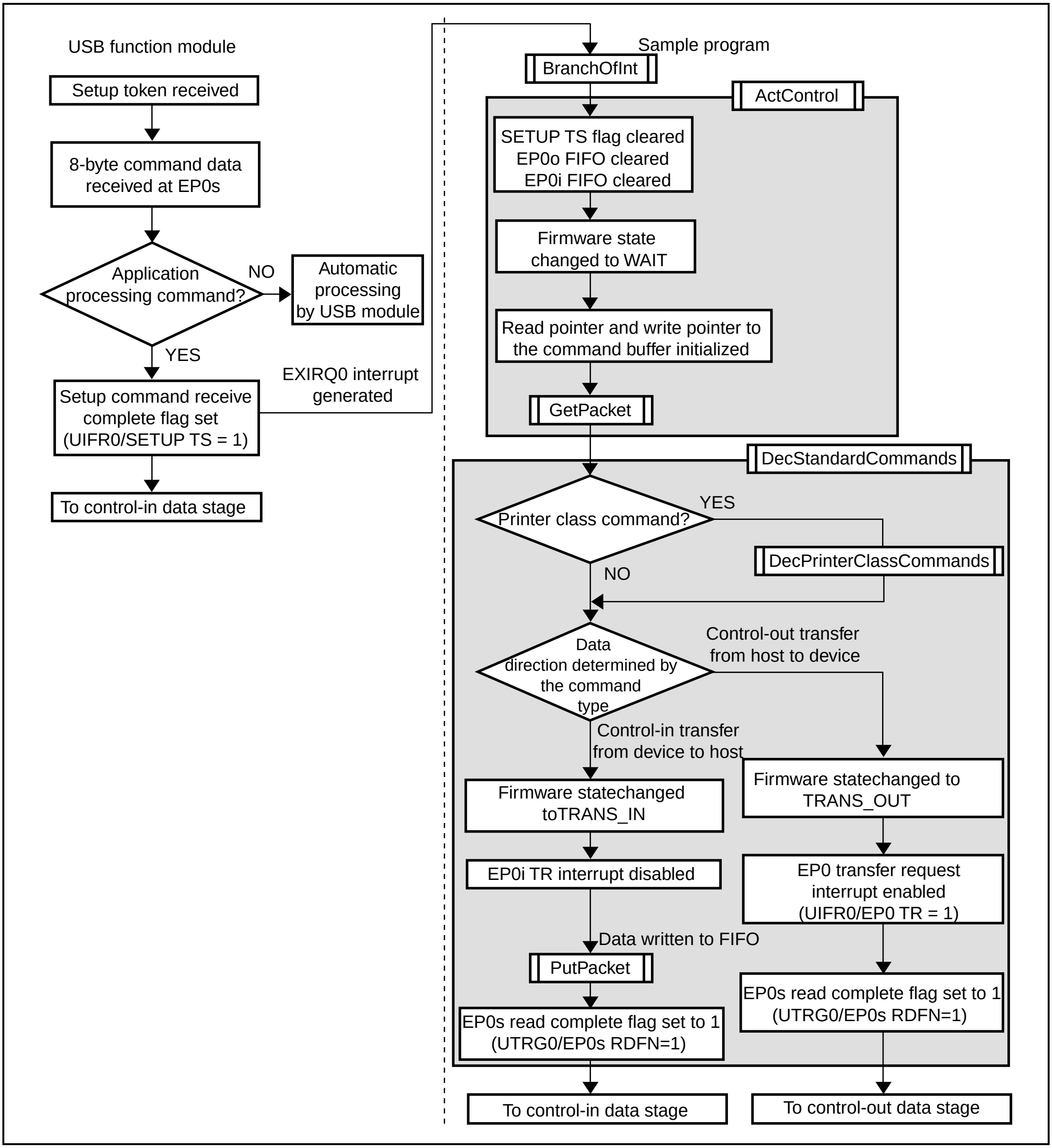
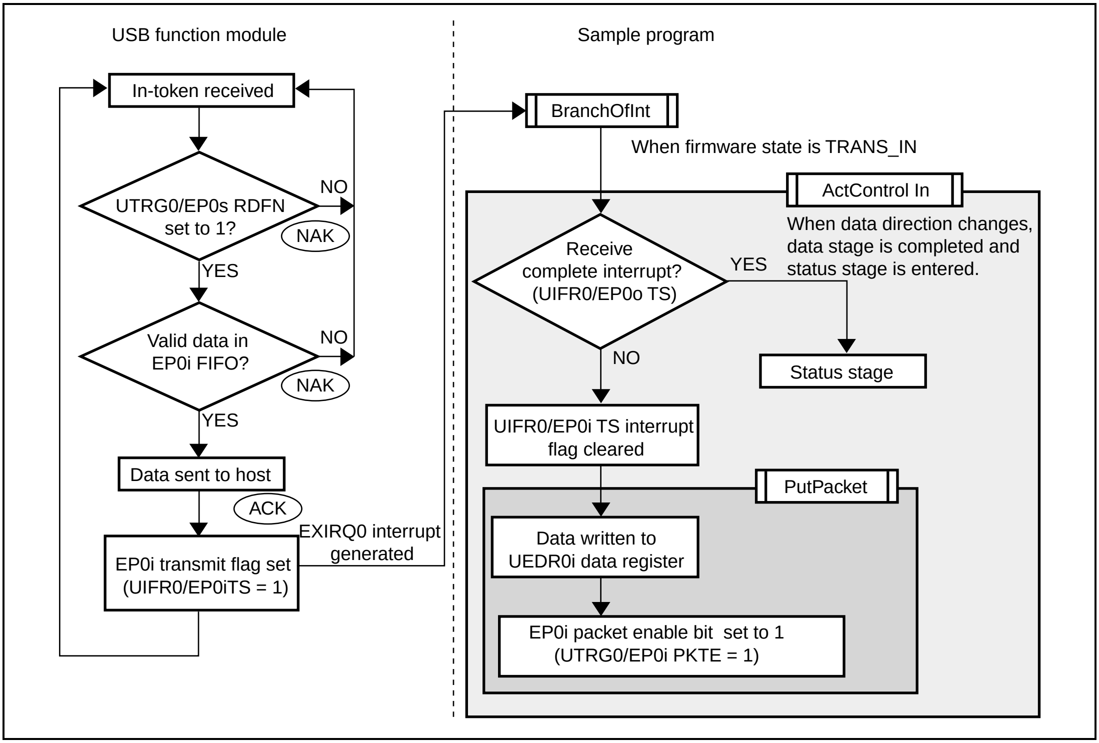

USB function module
Setup token received
8-byte command data received at EP0s
Application processing command?
NO
Automatic processing by USB module
YES
Setup command receive complete flag set (UIFR0/SETUP TS = 1)
EXIRQ0 interrupt generated
To control-in data stage
Sample program
BranchOfInt
ActControl
SETUP TS flag cleared
EP0o FIFO cleared
EP0i FIFO cleared
Firmware state changed to WAIT
Read pointer and write pointer to the command buffer initialized
GetPacket
DecStandardCommands
Printer class command?
YES
DecPrinterClassCommands
NO
Data direction determined by the command type
Control-out transfer from host to device
Control-in transfer from device to host
Firmware statechanged toTRANS_IN
Firmware statechanged to TRANS_OUT
EP0i TR interrupt disabled
EP0 transfer request interrupt enabled (UIFR0/EP0 TR = 1)
Data written to FIFO
PutPacket
EP0s read complete flag set to 1 (UTRG0/EP0s RDFN=1)
EP0s read complete flag set to 1 (UTRG0/EP0s RDFN=1)
To control-in data stage
To control-out data stage

Figure 5.9 Setup Stage

5.6.2 Data Stage

In the data stage, the host and function module exchange data. The firmware state becomes TRANS_IN for control-in transfers, and TRANS_OUT for control-out transfers, according to the result of decoding of the command in the setup stage. Figures 5.10 and 5.11 show the operation of the sample program in the data stage of control transfer



USB function module
Sample program
In-token received
UTRG0/EP0s RDFN set to 1?
NO
NAK
YES
Valid data in EP0i FIFO?
NO
NAK
YES
Data sent to host
ACK
EP0i transmit flag set (UIFR0/EP0iTS = 1)
EXIRQ0 interrupt generated
BranchOfInt
When firmware state is TRANS_IN
ActControl In
When data direction changes, data stage is completed and status stage is entered.
Receive complete interrupt? (UIFR0/EP0o TS)
YES
Status stage
NO
UIFR0/EP0i TS interrupt flag cleared
PutPacket
Data written to UEDR0i data register
EP0i packet enable bit set to 1 (UTRG0/EP0i PKTE = 1)

Figure 5.10 Data Stage (Control-In Transfer)

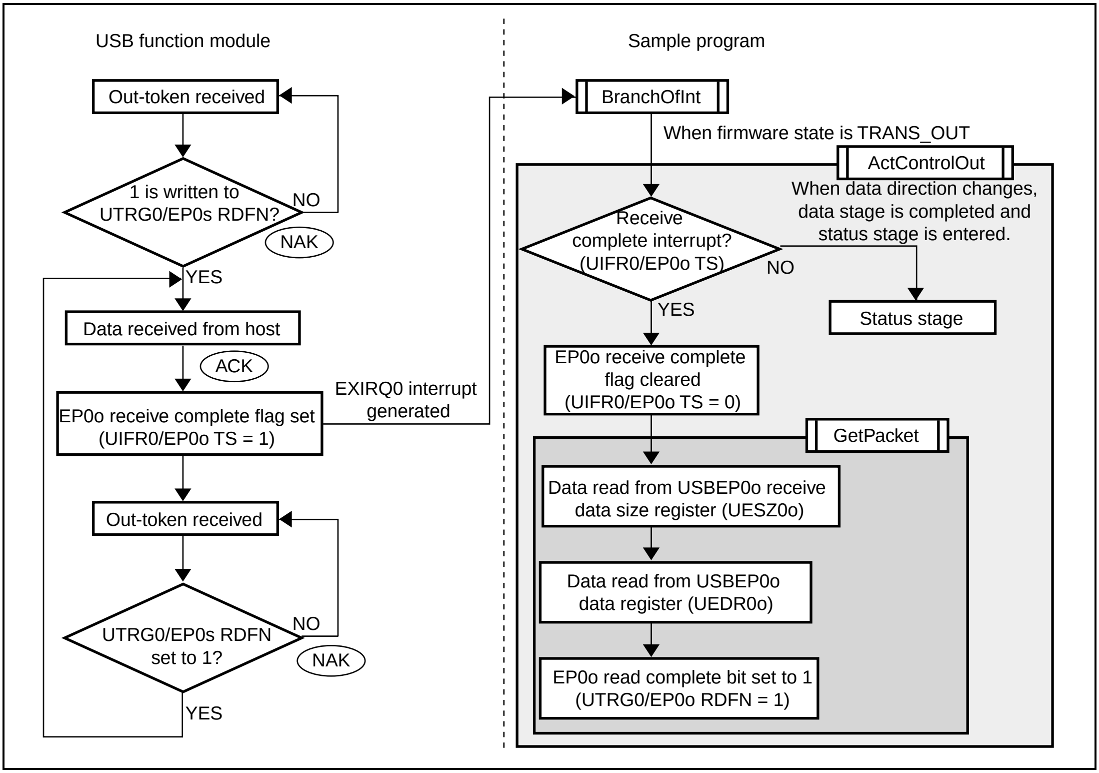

USB function module
Sample program
Out-token received
1 is written to UTRG0/EP0s RDFN?
NO
NAK
YES
Data received from host
ACK
EP0o receive complete flag set (UIFR0/EP0o TS = 1)
EXIRQ0 interrupt generated
Out-token received
UTRG0/EP0s RDFN set to 1?
NO
NAK
YES
BranchOfInt
When firmware state is TRANS_OUT
ActControlOut
Receive complete interrupt? (UIFR0/EP0o TS)
NO
When data direction changes, data stage is completed and status stage is entered.
Status stage
YES
EP0o receive complete flag cleared (UIFR0/EP0o TS = 0)
GetPacket
Data read from USBEP0o receive data size register (UESZ0o)
Data read from USBEP0o data register (UEDR0o)
EP0o read complete bit set to 1 (UTRG0/EP0o RDFN = 1)

Figure 5.11 Data Stage (Control-Out Transfer)

5.6.3 Status Stage

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

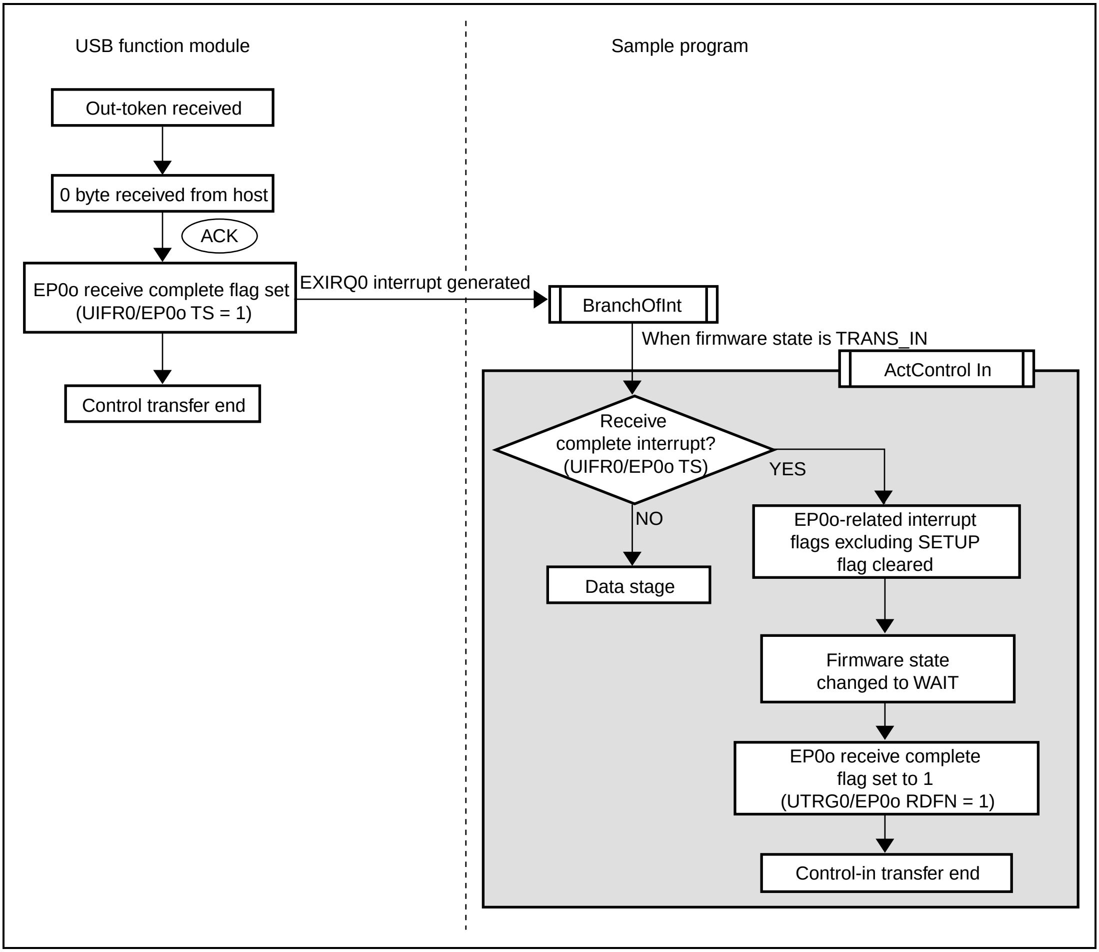

USB function module
Sample program
Out-token received
0 byte received from host
ACK
EP0o receive complete flag set (UIFR0/EP0o TS = 1)
EXIRQ0 interrupt generated
BranchOfInt
When firmware state is TRANS_IN
ActControl In
Control transfer end
Receive complete interrupt? (UIFR0/EP0o TS)
YES
NO
Data stage
EP0o-related interrupt flags excluding SETUP flag cleared
Firmware state changed to WAIT
EP0o receive complete flag set to 1 (UTRG0/EP0o RDFN = 1)
Control-in transfer end

Figure 5.12 Status Stage (Control-In Transfer)

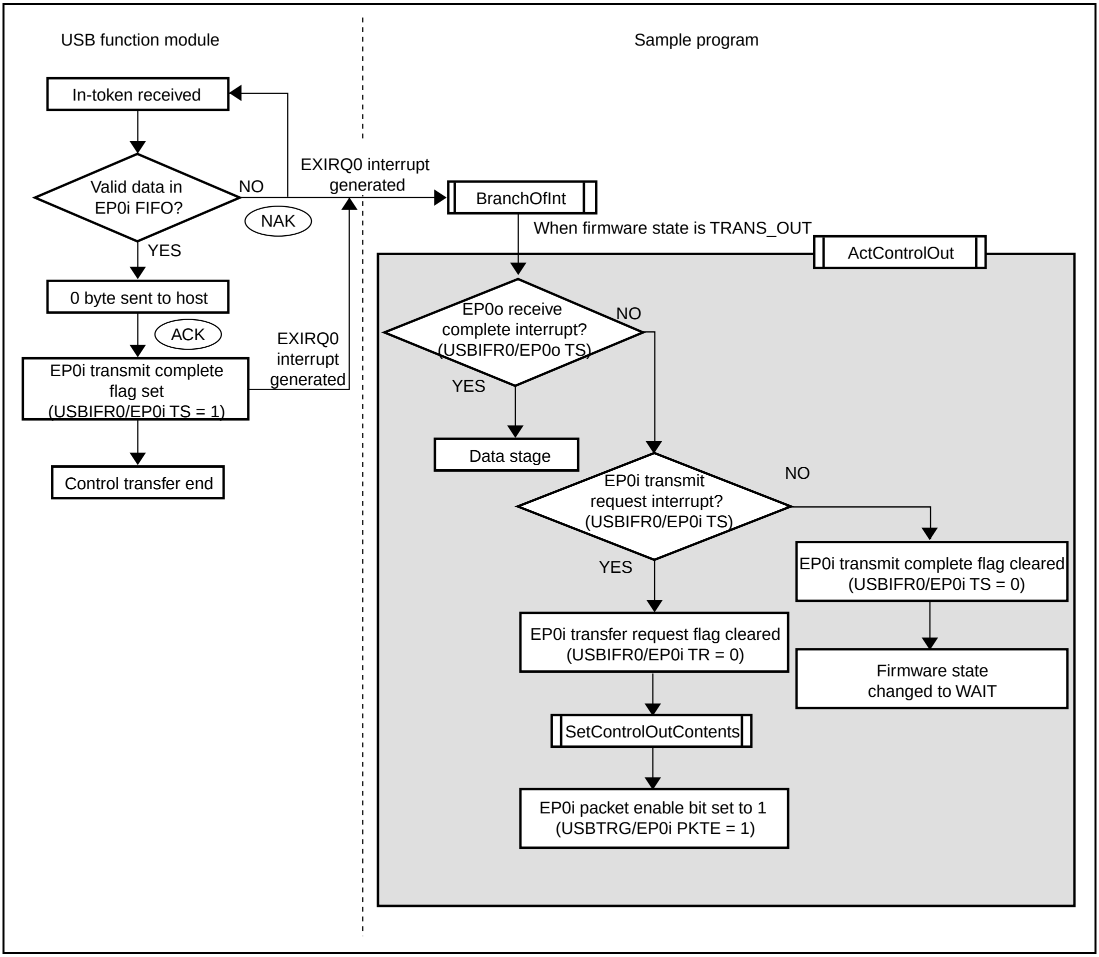

USB function module
Sample program
In-token received
Valid data in EP0i FIFO?
NO
NAK
EXIRQ0 interrupt generated
BranchOfInt
When firmware state is TRANS_OUT
ActControlOut
YES
0 byte sent to host
ACK
EP0i transmit complete flag set (USBIFR0/EP0i TS = 1)
EXIRQ0 interrupt generated
Control transfer end
EP0o receive complete interrupt? (USBIFR0/EP0o TS)
NO
YES
Data stage
EP0i transmit request interrupt? (USBIFR0/EP0i TS)
NO
YES
EP0i transmit complete flag cleared (USBIFR0/EP0i TS = 0)
Firmware state changed to WAIT
EP0i transfer request flag cleared (USBIFR0/EP0i TR = 0)
SetControlOutContents
EP0i packet enable bit set to 1 (USBTRG/EP0i PKTE = 1)

Figure 5.13 Status Stage (Control-Out Transfer)

5.7 Bulk Transfers

In bulk transfers, bits 0 to 2 of interrupt flag register 1 are used. Bulk transfers can also be divided into two types according to the direction of data transmission. (figure 5.14)

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

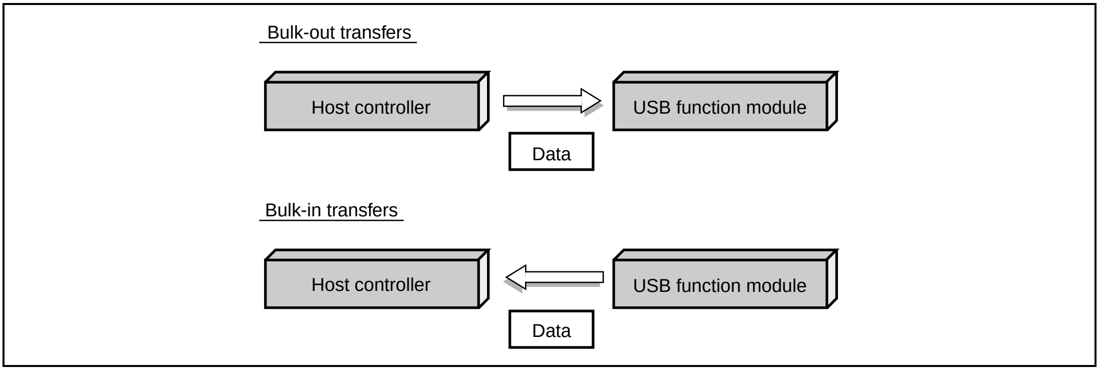


Figure 5.14 Bulk Transfers

5.7.1 Bulk-Out Transfers

The operation of the sample program in bulk-out transfers is shown in figure 5.15.

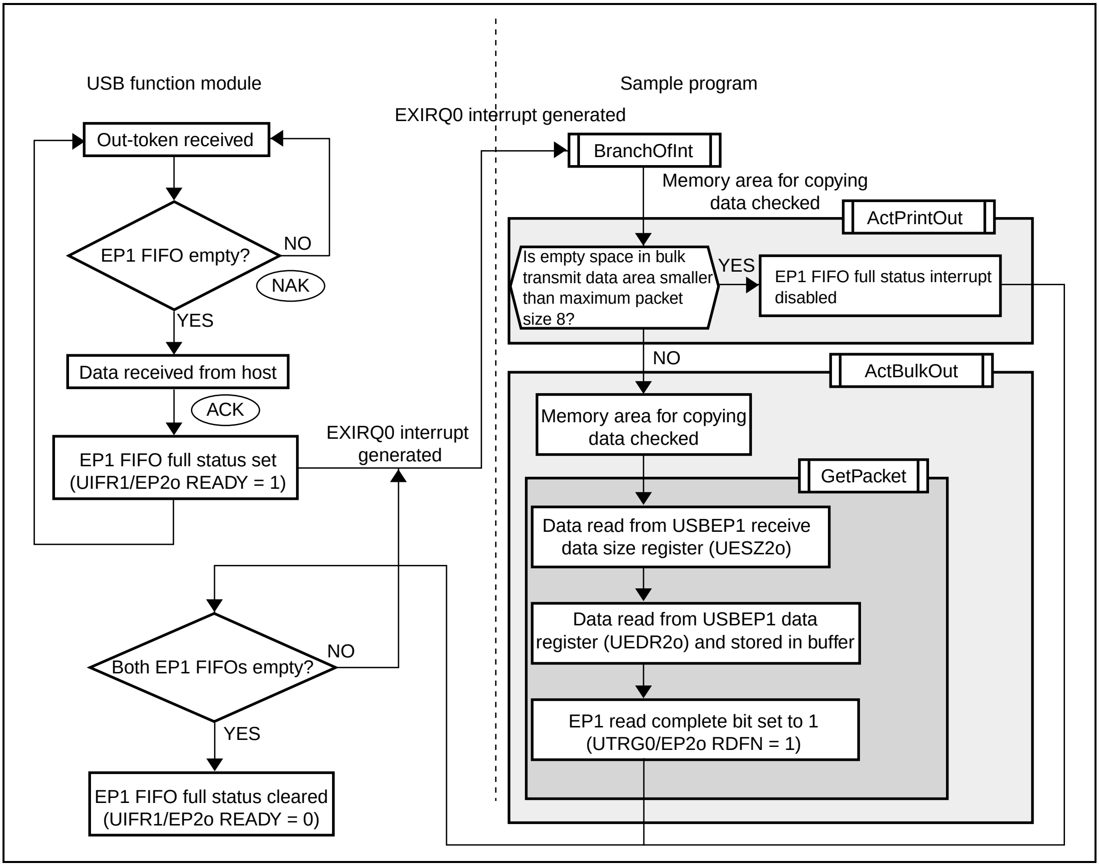

USB function module
Sample program
EXIRQ0 interrupt generated
Out-token received
EP1 FIFO empty?
NO
NAK
YES
Data received from host
ACK
EP1 FIFO full status set
(UIFR1/EP2o READY = 1)
EXIRQ0 interrupt generated
Both EP1 FIFOs empty?
NO
YES
EP1 FIFO full status cleared
(UIFR1/EP2o READY = 0)
BranchOfInt
Memory area for copying data checked
ActPrintOut
Is empty space in bulk transmit data area smaller than maximum packet size 8?
YES
EP1 FIFO full status interrupt disabled
NO
ActBulkOut
Memory area for copying data checked
GetPacket
Data read from USBEP1 receive data size register (UESZ2o)
Data read from USBEP1 data register (UEDR2o) and stored in buffer
EP1 read complete bit set to 1
(UTRG0/EP2o RDFN = 1)

Figure 5.15 Bulk-Out Transfers

5.7.2 Bulk-in Transfers

Figure 5.16 shows the operation of the sample program in bulk-in transfers.

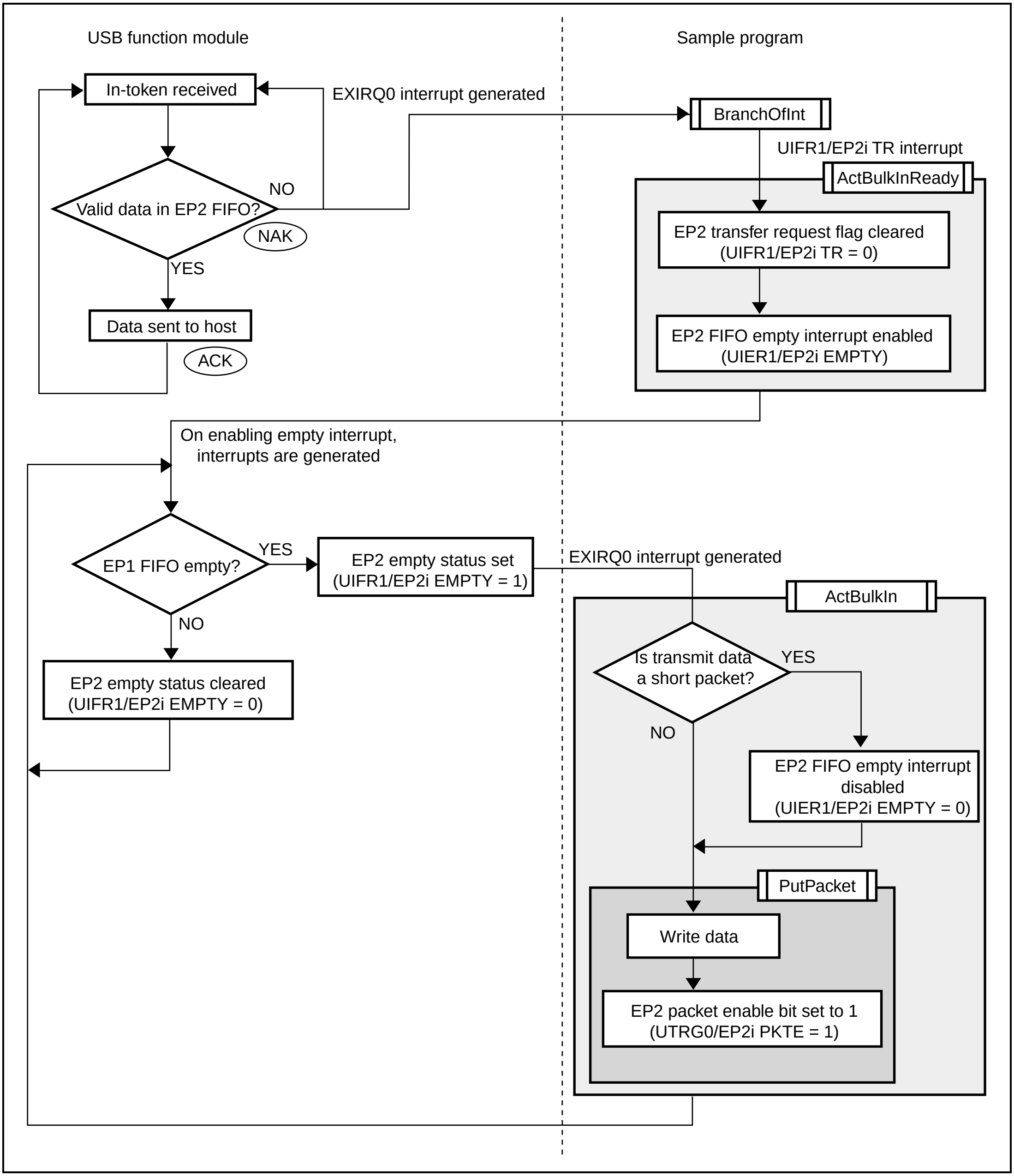


Figure 5.16 Bulk-In Transfers

Section 6 Analyzer Data

In this chapter, we look at how measurement is carried out with the USB Inspector, a USB protocol analyzer made 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 bulk-out transport in printing out as examples. For more detailed information on packets, see section 2.6.1.

Note: The Packet # found in front of each packet is the packet number used when measuring.

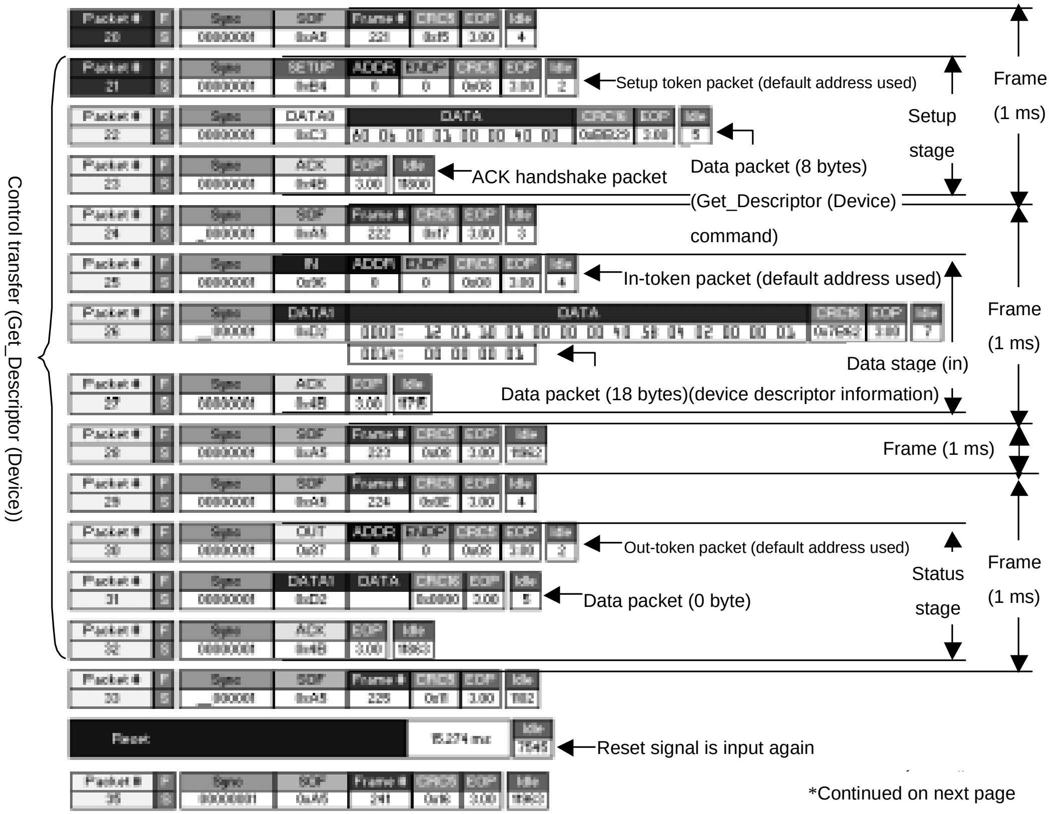
The Idle found at the end of each packet indicates the idle between packets (see sections 2.2 and 2.6).

6.1 Control Transfer When a Device Is Connected

Figure 6.1 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 (the device is ready for being used (configuration state). For details on the state transitions, see section 2.7.1.

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

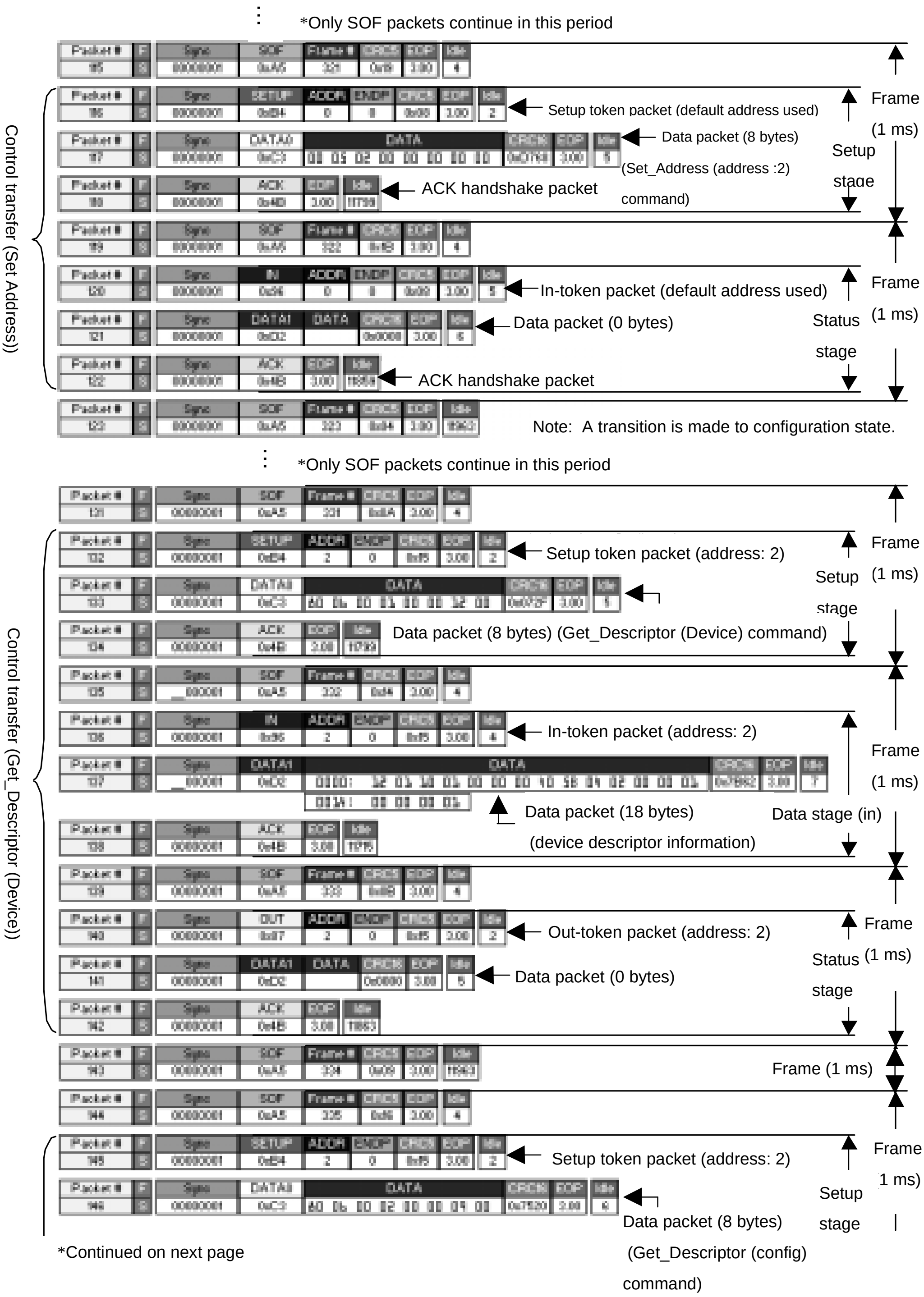
Reset signal. A transition is made from power-on state to default state.



```
Packet # 20 | F S | Sync 00000001 | SOF 0xA5 | Frame # 221 | CRC5 0xf5 | EOP 3.00 | Idle 4
Packet # 21 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 2
Packet # 22 | F S | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 01 00 00 40 00 | CRC16 0xBB29 | EOP 3.00 | Idle 5
Packet # 23 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11800
Packet # 24 | F S | Sync _0000001 | SOF 0xA5 | Frame # 222 | CRC5 0x17 | EOP 3.00 | Idle 3
Packet # 25 | F S | Sync 00000001 | IN 0x96 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 4
Packet # 26 | F S | Sync _000001 | DATA1 0xD2 | DATA 0001: 12 01 10 01 00 00 00 40 5B 04 02 00 00 01 | CRC16 0x7E92 | EOP 3.00 | Idle 7
                                             0014: 01 02 00 01
Packet # 27 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11795
Packet # 28 | F S | Sync 00000001 | SOF 0xA5 | Frame # 223 | CRC5 0x08 | EOP 3.00 | Idle 11862
Packet # 29 | F S | Sync 00000001 | SOF 0xA5 | Frame # 224 | CRC5 0x0E | EOP 3.00 | Idle 4
Packet # 30 | F S | Sync 00000001 | OUT 0x87 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 2
Packet # 31 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 5
Packet # 32 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11863
Packet # 33 | F S | Sync _000001 | SOF 0xA5 | Frame # 225 | CRC5 0x11 | EOP 3.00 | Idle 1182
Reset 15.274 ms | Idle 7545
Packet # 35 | F S | Sync 00000001 | SOF 0xA5 | Frame # 241 | CRC5 0x16 | EOP 3.00 | Idle 11863
```

Control transfer (Get_Descriptor (Device))
Setup token packet (default address used)
Data packet (8 bytes)
(Get_Descriptor (Device)
command)
ACK handshake packet
Setup
stage
Frame
(1 ms)
In-token packet (default address used)
Data packet (18 bytes)(device descriptor information)
Data stage (in)
Frame
(1 ms)
Frame (1 ms)
Out-token packet (default address used)
Data packet (0 byte)
Status
stage
Frame
(1 ms)
Reset signal is input again

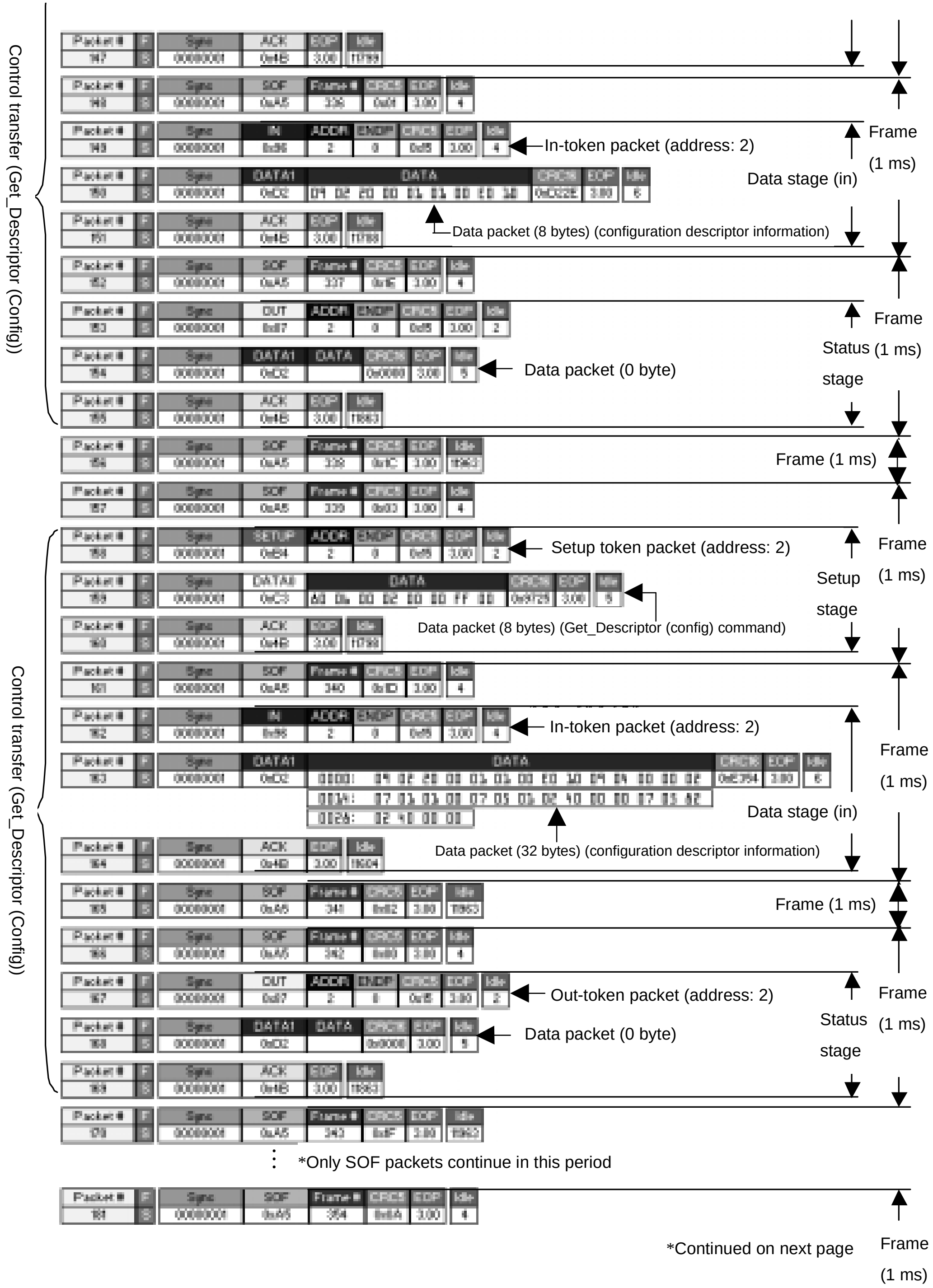
*Continued on next page

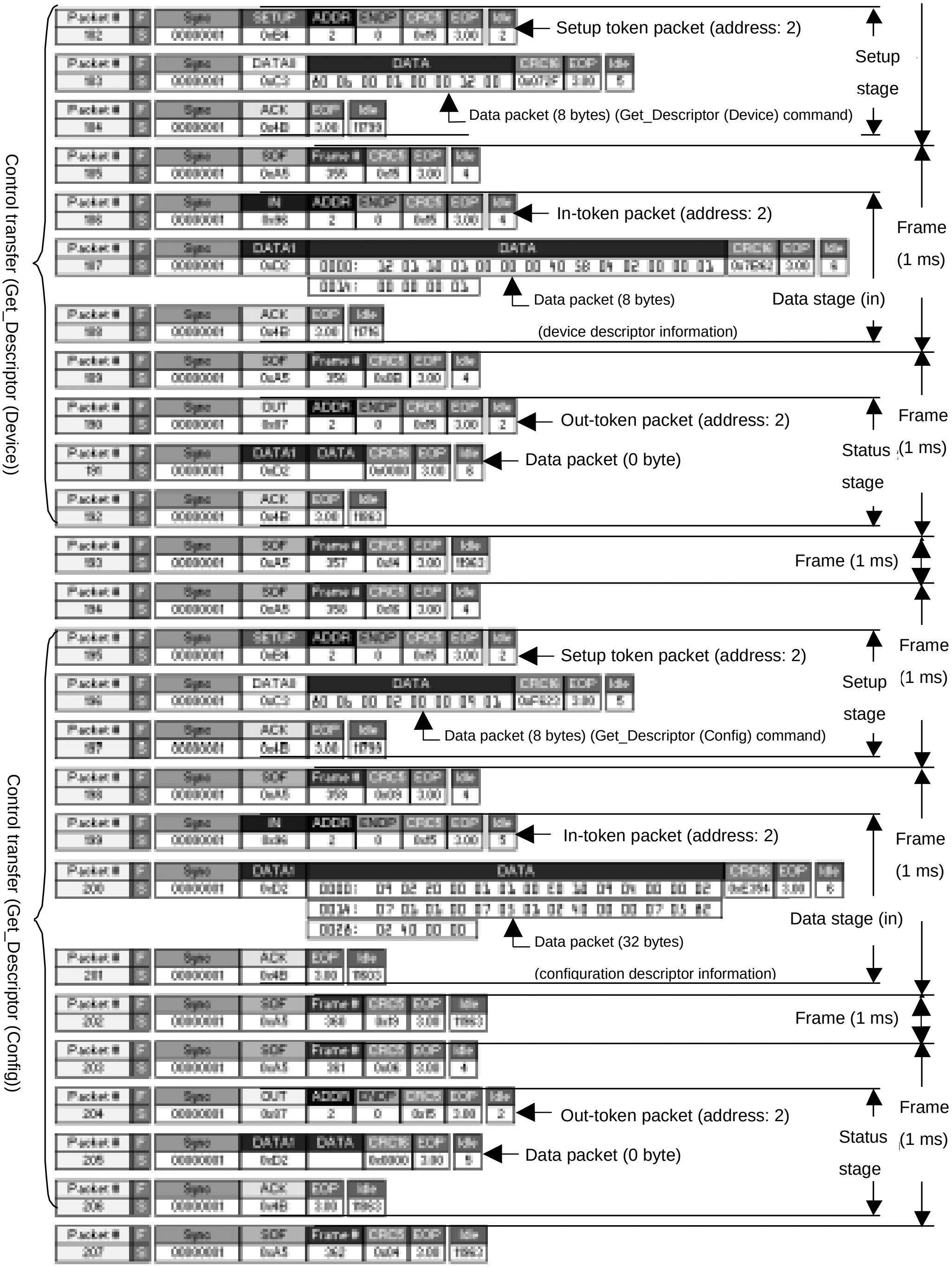


⋮
*Only SOF packets continue in this period
Setup token packet (default address used)
Data packet (8 bytes)
(Set_Address (address :2)
command)
ACK handshake packet
In-token packet (default address used)
Data packet (0 bytes)
ACK handshake packet
Frame
(1 ms)
Setup
stage
Frame
(1 ms)
Status
stage
Control transfer (Set Address))
Note: A transition is made to configuration state.
⋮
*Only SOF packets continue in this period
Setup token packet (address: 2)
Data packet (8 bytes) (Get_Descriptor (Device) command)
In-token packet (address: 2)
Data packet (18 bytes)
(device descriptor information)
Out-token packet (address: 2)
Data packet (0 bytes)
Setup token packet (address: 2)
Data packet (8 bytes)
(Get_Descriptor (config)
command)
Frame
(1 ms)
Setup
stage
Frame
(1 ms)
Data stage (in)
Frame
(1 ms)
Status
stage
Frame (1 ms)
Frame
1 ms)
Setup
stage
Control transfer (Get_Descriptor (Device))

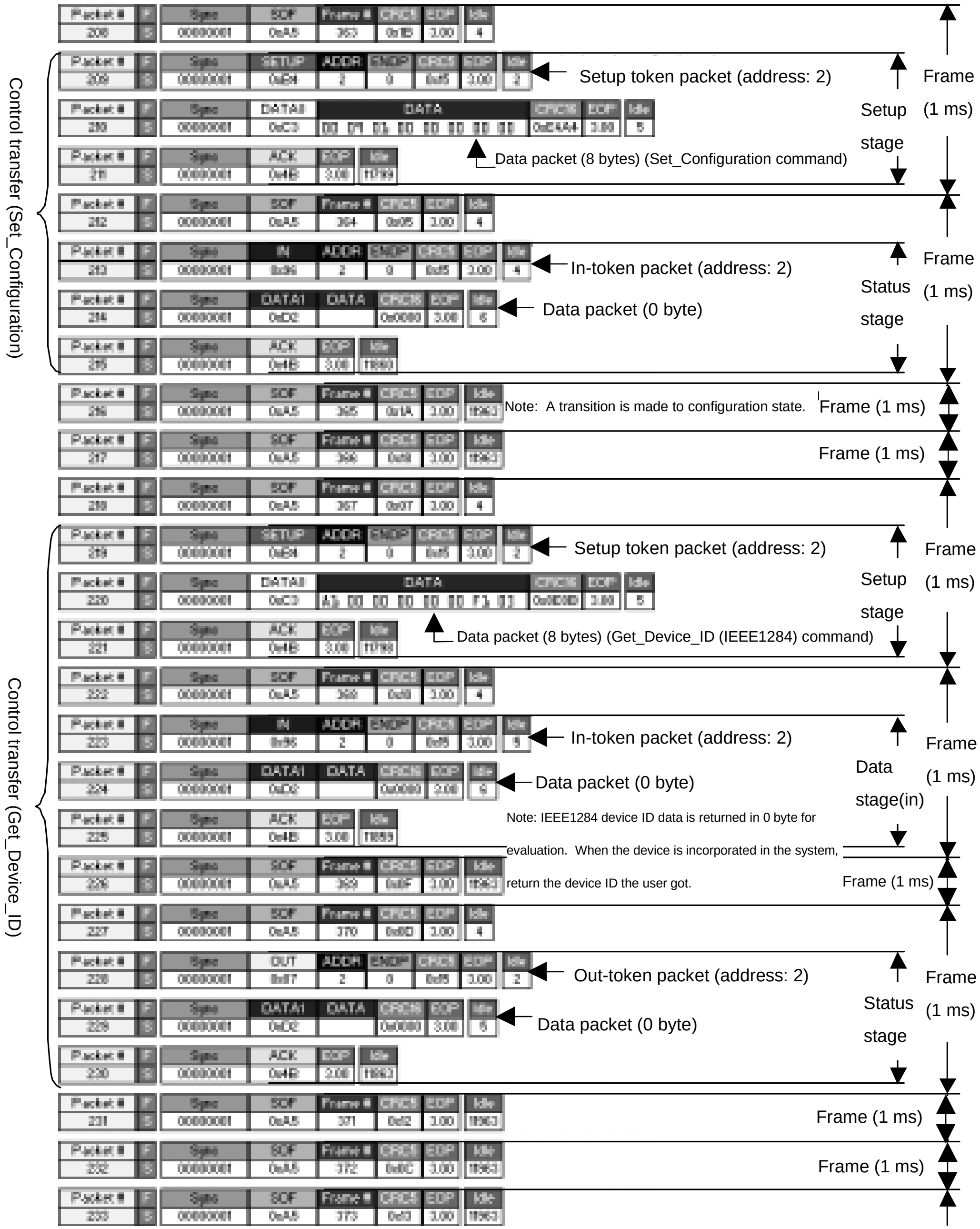
```
Packet # 115 | F S | Sync 00000001 | SOF 0xA5 | Frame # 321 | CRC5 0x19 | EOP 3.00 | Idle 4
Packet # 116 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 2
Packet # 117 | F S | Sync 00000001 | DATA0 0xC3 | DATA 00 05 02 00 00 00 00 00 | CRC16 0xD768 | EOP 3.00 | Idle 5
Packet # 118 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11799
Packet # 119 | F S | Sync 00000001 | SOF 0xA5 | Frame # 322 | CRC5 0x0B | EOP 3.00 | Idle 4
Packet # 120 | F S | Sync 00000001 | IN 0x96 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 5
Packet # 121 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 6
Packet # 122 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11858
Packet # 123 | F S | Sync 00000001 | SOF 0xA5 | Frame # 323 | CRC5 0x04 | EOP 3.00 | Idle 11962

Packet # 131 | F S | Sync 00000001 | SOF 0xA5 | Frame # 331 | CRC5 0x1A | EOP 3.00 | Idle 4
Packet # 132 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Packet # 133 | F S | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 01 00 00 12 00 | CRC16 0x072F | EOP 3.00 | Idle 5
Packet # 134 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11799
Packet # 135 | F S | Sync __000001 | SOF 0xA5 | Frame # 332 | CRC5 0x14 | EOP 3.00 | Idle 4
Packet # 136 | F S | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 4
Packet # 137 | F S | Sync __000001 | DATA1 0xD2 | DATA 0000: 12 01 10 01 00 00 00 40 56 04 02 00 00 01
                                                     0014: 00 01 00 01 | CRC16 0x7B82 | EOP 3.00 | Idle 7
Packet # 138 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11915
Packet # 139 | F S | Sync 00000001 | SOF 0xA5 | Frame # 333 | CRC5 0x0B | EOP 3.00 | Idle 4
Packet # 140 | F S | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Packet # 141 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 5
Packet # 142 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11863
Packet # 143 | F S | Sync 00000001 | SOF 0xA5 | Frame # 334 | CRC5 0x09 | EOP 3.00 | Idle 11963
Packet # 144 | F S | Sync 00000001 | SOF 0xA5 | Frame # 335 | CRC5 0x16 | EOP 3.00 | Idle 4
Packet # 145 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Packet # 146 | F S | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 02 00 00 09 00 | CRC16 0x7520 | EOP 3.00 | Idle 6
```

*Continued on next page


Control transfer (Get_Descriptor (Config))
In-token packet (address: 2)
Data stage (in)
Frame (1 ms)
Data packet (8 bytes) (configuration descriptor information)
Frame (1 ms)
Status stage
Frame (1 ms)
Data packet (0 byte)
Frame (1 ms)
Control transfer (Get_Descriptor (Config))
Setup token packet (address: 2)
Setup stage
Frame (1 ms)
Data packet (8 bytes) (Get_Descriptor (config) command)
In-token packet (address: 2)
Frame (1 ms)
Data stage (in)
Data packet (32 bytes) (configuration descriptor information)
Frame (1 ms)
Out-token packet (address: 2)
Status stage
Frame (1 ms)
Data packet (0 byte)
*Only SOF packets continue in this period
*Continued on next page
Frame (1 ms)


Control transfer (Get_Descriptor (Device))
Setup token packet (address: 2)
Setup stage
Data packet (8 bytes) (Get_Descriptor (Device) command)
Frame (1 ms)
In-token packet (address: 2)
Data packet (8 bytes)
(device descriptor information)
Data stage (in)
Out-token packet (address: 2)
Data packet (0 byte)
Frame (1 ms)
Status stage
Frame (1 ms)
Control transfer (Get_Descriptor (Config))
Setup token packet (address: 2)
Setup stage
Frame (1 ms)
Data packet (8 bytes) (Get_Descriptor (Config) command)
In-token packet (address: 2)
Frame (1 ms)
Data packet (32 bytes)
(configuration descriptor information)
Data stage (in)
Frame (1 ms)
Out-token packet (address: 2)
Data packet (0 byte)
Status stage
Frame (1 ms)

*Continued on next page


Packet # 208 | F S | Sync 00000001 | SOF 0xA5 | Frame # 363 | CRC5 0x1B | EOP 3.00 | Idle 4
Packet # 209 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Setup token packet (address: 2)
Packet # 210 | F S | Sync 00000001 | DATA0 0xC3 | DATA 00 09 01 00 00 00 00 00 | CRC16 0xE4A4 | EOP 3.00 | Idle 5
Packet # 211 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11769
Data packet (8 bytes) (Set_Configuration command)
Setup
stage
Frame
(1 ms)
Packet # 212 | F S | Sync 00000001 | SOF 0xA5 | Frame # 364 | CRC5 0x05 | EOP 3.00 | Idle 4
Packet # 213 | F S | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 4
In-token packet (address: 2)
Packet # 214 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 6
Data packet (0 byte)
Packet # 215 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11660
Status
stage
Frame
(1 ms)
Control transfer (Set_Configuration)
Packet # 216 | F S | Sync 00000001 | SOF 0xA5 | Frame # 365 | CRC5 0x1A | EOP 3.00 | Idle 11963
Note: A transition is made to configuration state.
Frame (1 ms)
Packet # 217 | F S | Sync 00000001 | SOF 0xA5 | Frame # 366 | CRC5 0x19 | EOP 3.00 | Idle 11963
Frame (1 ms)
Packet # 218 | F S | Sync 00000001 | SOF 0xA5 | Frame # 367 | CRC5 0x07 | EOP 3.00 | Idle 4
Packet # 219 | F S | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Setup token packet (address: 2)
Packet # 220 | F S | Sync 00000001 | DATA0 0xC3 | DATA A1 00 00 00 00 00 F1 03 | CRC16 0x0D0D | EOP 3.00 | Idle 5
Packet # 221 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11769
Data packet (8 bytes) (Get_Device_ID (IEEE1284) command)
Setup
stage
Frame
(1 ms)
Packet # 222 | F S | Sync 00000001 | SOF 0xA5 | Frame # 368 | CRC5 0x18 | EOP 3.00 | Idle 4
Packet # 223 | F S | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 5
In-token packet (address: 2)
Packet # 224 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 6
Data packet (0 byte)
Packet # 225 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11859
Data
stage(in)
Frame
(1 ms)
Note: IEEE1284 device ID data is returned in 0 byte for evaluation. When the device is incorporated in the system, return the device ID the user got.
Packet # 226 | F S | Sync 00000001 | SOF 0xA5 | Frame # 369 | CRC5 0x0F | EOP 3.00 | Idle 11963
Frame (1 ms)
Packet # 227 | F S | Sync 00000001 | SOF 0xA5 | Frame # 370 | CRC5 0x0D | EOP 3.00 | Idle 4
Packet # 228 | F S | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Out-token packet (address: 2)
Packet # 229 | F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 5
Data packet (0 byte)
Packet # 230 | F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11963
Status
stage
Frame
(1 ms)
Control transfer (Get_Device_ID)
Packet # 231 | F S | Sync 00000001 | SOF 0xA5 | Frame # 371 | CRC5 0x02 | EOP 3.00 | Idle 11963
Frame (1 ms)
Packet # 232 | F S | Sync 00000001 | SOF 0xA5 | Frame # 372 | CRC5 0x0C | EOP 3.00 | Idle 11963
Frame (1 ms)
Packet # 233 | F S | Sync 00000001 | SOF 0xA5 | Frame # 373 | CRC5 0x03 | EOP 3.00 | Idle 11963

Note: The stationary state continues until a bulk transfer is performed.

Figure 6.1 Control Transfer When a Device is Connected

6.2 Bulk-Out Transport for Printing Out (For the bulk-out transport, refer to section 2.6.3.)

Figure 6.2 shows the measurement results when the bulk-out transport (printing out) is performed from the host controller to this device.

For each transfer, the PID of data packets is toggled like DATA0 → DATA1 → DATA0.

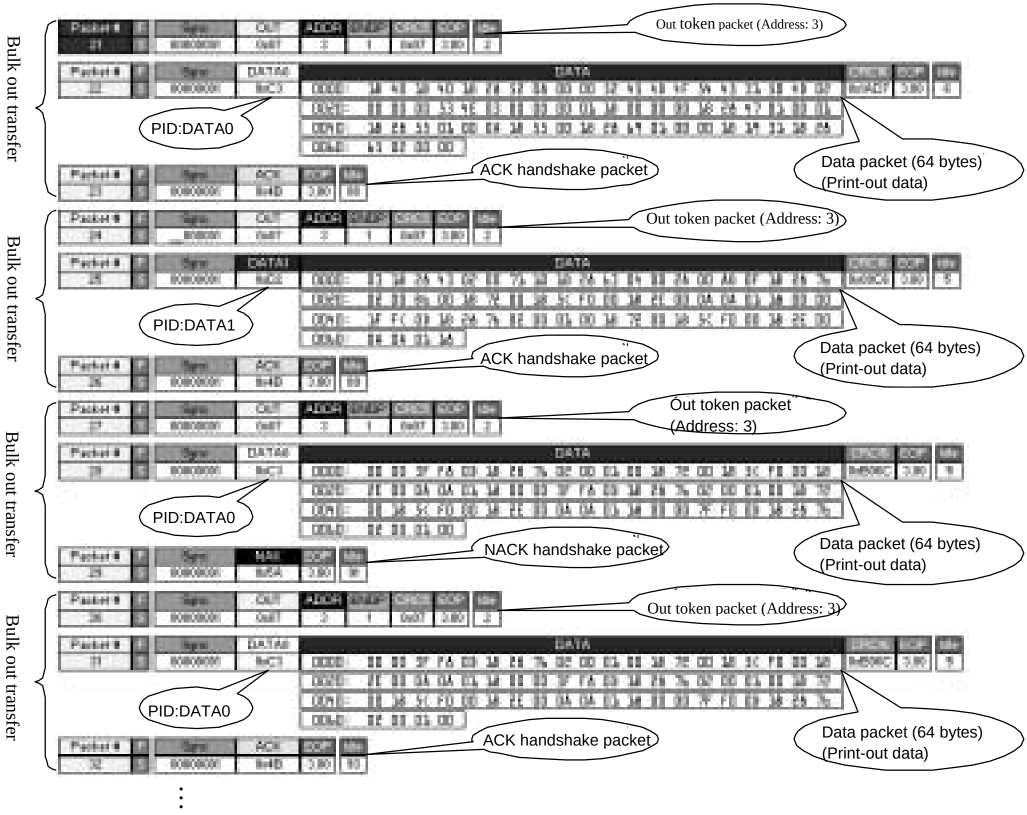


Figure 6.2 Bulk-Out Transport for Printing Out

H8S/2215 USB Function Module

Application Notes

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