To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

RENESAS

Notice

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

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

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

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

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

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

Application Note

16

H8S/2218 USB Function Module Human Interface Devices (HID) Class

Application Note

Renesas 16-Bit Single-Chip Microcomputer

H8S Family / H8S/2200 Series

Renesas Electronics
www.renesas.com

Rev.1.00 2003.10

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corp. 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 Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.

Preface

These application notes describe the HID class firmware that uses the USB Function Module in the H8S/2218. They are provided to be used as a reference when the user creates USB Function Module firmware.

These application notes describe a system configuration example for HID class communications based on the USB Function Module, and do not guarantee the contents of the configuration.

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

[Related manuals]

- Universal Serial Bus Specification Revision 1.1
- Universal Serial Bus Device Class Definition for Human Interface Devices (HID)
- H8S/2218 Group, H8S/2212 Group Hardware Manual
- H8S/2218 Solution Engine CPU Board (MS2218CP01) Instruction Manual

[Caution] The sample programs described in these application notes do not include firmware related to bulk transfer, which is a USB transfer type. When using this transfer type (see section 14.5.6 to section 14.5.7 of the H8S/2218 Group, H8S/2212 Group Hardware Manual), the user needs to create the programs for it.

Also, the hardware specifications of the H8S/2218 and H8S/2218 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/2218 Group, H8S/2212 Group Hardware Manual and the H8S/2218 Solution Engine Instruction Manual.

Contents

RENESAS

Section 1 Overview

This application note describes how to use the USB Function Module that is built into the H8S/2218, and contain examples of firmware programs.

The features of the USB Function Module contained in the H8S/2218 are listed below.

- An on-chip UDC (USB Device Controller) conforming to USB 1.1
- Automatic processing of USB protocol
- Automatic processing of USB standard commands for endpoint 0 (some commands need to be processed through the firmware)
- Full-speed (12 Mbps) transfer supported
- Various interrupt signals needed for USB transmission and reception are generated.
- An on-chip bus transceiver

Endpoint Configurations

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

Figure 1.1 shows an example of a system configuration.

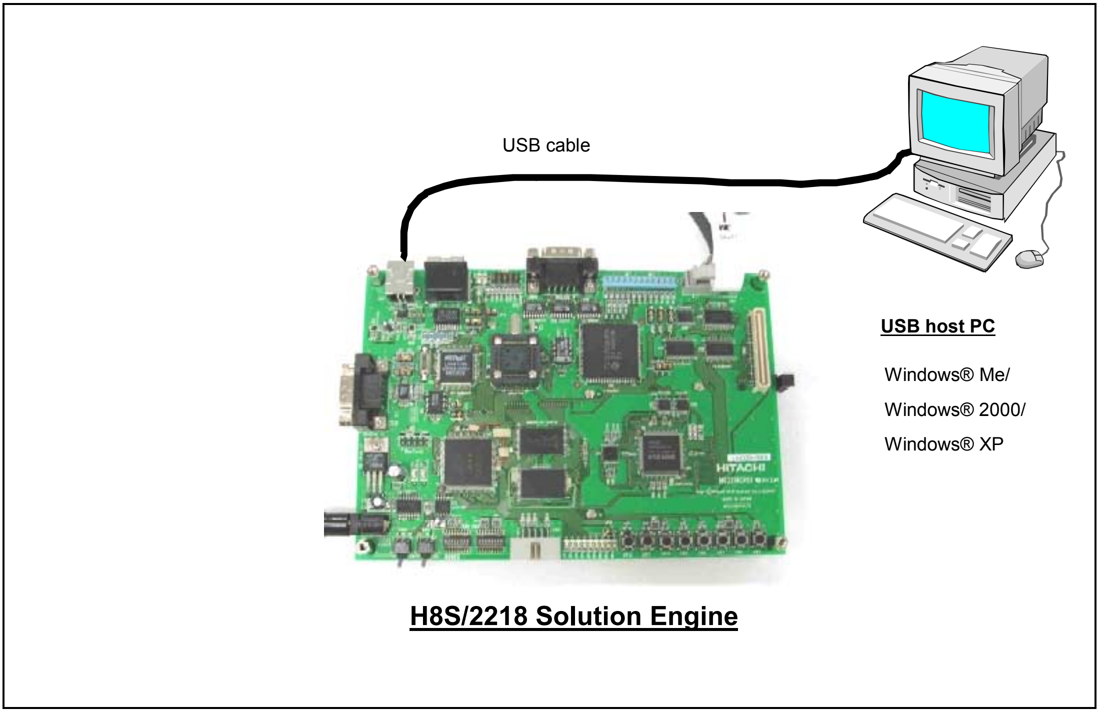


Figure 1.1 System Configuration Example

This system is configured of the H8S/2218 Solution Engine manufactured by Hitachi ULSI Systems Co., Ltd. (hereafter referred to as the MS2218CP) and a PC containing Windows® Me/ Windows® 2000/Windows® XP operating system.

This system is an HID class firmware that automatically generates pseudo mouse data on the MS2218CP board and outputs the mouse data (hereafter called the HID data) to the host PC through the USB.

It is also possible to use the USB HID class device driver that comes as an accessory with the operating systems listed above.

This system offers the following features.

1. The sample program can be used to evaluate the USB module of the H8S/2218 quickly.
2. The sample program supports USB control transfer and interrupt transfer.
3. An E10A can be used, enabling efficient debugging.

Section 2 Overview of the USB Human Interface Devices (HID) Class

This section describes the USB Human Interface Devices (HID) Class.

We hope that it will provide a convenient reference for use when developing USB HID class devices. For more detailed information on standards, please see the following:

- Device Class Definition for Human Interface Devices (HID) Version 1.11
- HID Usage Tables Version 1.11

2.1 HID Class

USB HID class is a class of standards that apply to devices through which humans operate PCs. Typical examples include mouse devices, keyboards, and joysticks.

To notify the host PC of this class of function, the bInterfaceClass filed of the Interface descriptor must be 0x03.

2.2 Subclass Code

Subclasses were intended to be used to identify the specific protocols of different HID class devices. However, as there are many types of devices used by humans, subclass protocol definitions are impractical, and subclasses are not used to define most protocols in the HID class. Instead, the protocol is identified by the Report descriptor in HID class devices.

As for BIOS-support devices (boot devices), a simple method to identify the protocol is needed. For this purpose, subclasses are used to indicate devices that support the predefined protocol (boot protocol) for mouse devices or keyboards (that is, devices that can be used for boot devices).

To notify the host PC that the device supports the boot protocol, the bInterfaceSubClass filed of the Interface descriptor must be 0x01.

2.3 Protocol Code

When a device supports the boot protocol (subclass code other than 0), a protocol code is used to indicate the device type. The protocol code is 0x01 for a keyboard, and 0x02 for a mouse. Specifying the device type by the protocol code indicates that the device can use the protocol for the device type.

To notify the host PC of the device type, the bInterfaceProtocol filed of the Interface descriptor must be a value corresponding to the device type.

2.4 Descriptors for HID Class

HID class function devices need an HID descriptor, a Report descriptor, and a Physical descriptor (optional) in addition to descriptor information that other USB function devices need. Figure 2.1 shows the HID device descriptor configuration.

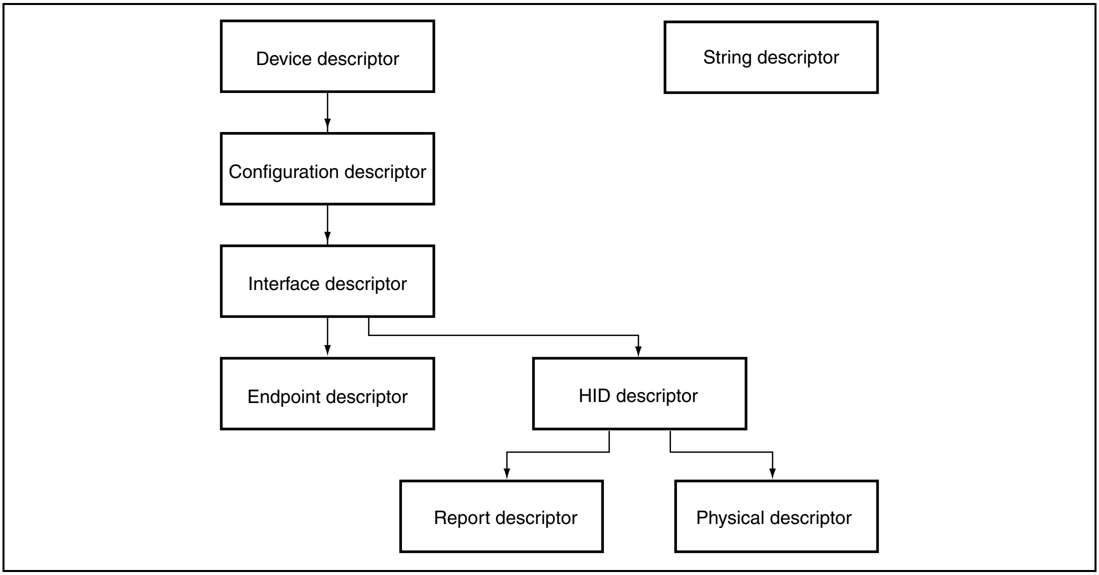


Figure 2.1 Descriptor Configuration

2.5 HID Descriptor

The HID descriptor combines the Report descriptor and Physical descriptor (optional). Table 2.1 shows the format of the HID descriptor.

Table 2.1 HID Descriptor

Field	Size (bytes)	Description
bLength	1	Descriptor size (fixed to 0x09)
bDescriptorType	1	Descriptor type (fixed to 0x21)
bcdHID	2	HID version in BCD
bCountryCode	1	Country ID for devices specific to a particular country (0 unless necessary)
bNumDescriptors	1	Number of class descriptors
bDescriptorType	1	Type of class descriptor (0x22 for HIDREPORT)
wDescriptorLength	2	Size of Report descriptor

2.6 Report Descriptor

The Report descriptor specifies the format of data to be transferred between the host PC and the device. Unlike other descriptors, the Report descriptor has no standardized format, but the length and contents of the Report descriptor vary depending on the device's report or the number of data fields required for the device's report.

The Report descriptor consists of items that provide information about the device. There are two types of items, short and long items. The following describes the short item.

Bits:	39 to 8	7654	32	10
Parts:	[data]	bTag	bType	bSize
Byte:	4 to 1		0	

bType and bSize indicate the following meaning.

item type	
bType Value	Type
00	Main
01	Global
10	Local
11	Reserve

item size	
bSize Value	Data Size
00	0 byte
01	1 byte
10	2 bytes
11	4 bytes

Figure 2.2 Report Descriptor Item

An item consists of four fields: data, item tag, item type, and itemSize. The item uses these fields to indicate the information.

There are three item types: Main, Global, and Local. The Main item type (defining or grouping the data fields in a Report descriptor) has five types of item tags, the Global item type (describing data) has 12, and the Local item type (defining the characteristics) has ten.

By combining these item tags, the Report descriptor specifies the format of data to be transferred between the host PC and the device.

2.6.1 Main Items

Table 2.2 shows five item tags for the Main item type.

Table 2.2 Item Tags for Main Item Type

Item Tag	bTag	bType	bSize	Description
Input	1000	00	nn	Describes information about data provided by one or more physical controls
Output	1001	00	nn	Defines output data field
Feature	1011	00	nn	Describes device configuration information that can be sent to the device
Collection	1010	00	nn	Starts collecting relations between two or more data item tags (Input, Output, or Feature)
End Collection	1100	00	nn	Ends collecting relations between two or more data item tags (Input, Output, or Feature) in response to Collection

Input Item Tag: The input item tag has eight parameters (data fields), which are set in 1-bit units, as shown in table 2.3.

Table 2.3 Input Item Tag Parameters

Bit	Value	Contents	Description
0	0	Data	The item reports data
	1	Constant	The item reports a constant
1	0	Array	The item reports an array data field
	1	Variable	The item reports a variable
2	0	Absolute	The item reports an absolute value
	1	Relative	The item reports a relative value from the last report
3	0	No Wrap	The value reported by the item does not roll over
	1	Wrap	The value reported by the item rolls over (for example, for a dial to output a value from 0 to 10, if dialing is continued, 0 is output after 10)
4	0	Linear	The item reports the state of the target control linearly
	1	Non Linear	The item processes raw data and does not report the state of the target linearly
5	0	Preferred State	The item has a state to which it returns when it is not controlled by the user
	1	No Preferred	The item does not have a state to which it returns when it is not controlled by the user
6	0	No Null position	The item has a state in which it does not send meaningful data
	1	Null state	The item does not have a state in which it does not send meaningful data
7	0	Reserved	Reserved
8	0	Bit Field	The item issues a bit field
	1	Buffered Bytes	The item issues a stream fixed to 1-byte size
9-31	0	Reserved	Reserved

Output and Feature Item Tags: The output and feature item tags have nine parameters (data fields), which are the same as the input item tag except bit 7, as shown in table 2.4.

Table 2.4 Output and Feature Item Tag Parameters

Bit	Value	Contents	Description
1-6	—	—	Same as the input item tag
7	0	Non Volatile	The item value cannot change with or without host interactions
	1	Volatile	The item value can change with or without host interactions
8-31	—	—	Same as the input item tag

Collection Item Tag: The collection item tag has eight parameters (data fields), which are set in one byte, as shown in table 2.5.

Table 2.5 Collection Item Tag Parameters

Value	Contents	Description
0x00	Physical	Used for data items collected into one. This is used for devices which need to associate correct or sensed data with a single point. It does not indicate that data comes from a single device such as a keyboard. It indicates that the device reports multiple sensor positions and data comes from different sensors.
0x01	Application	Identifies the Usage only used for the application level. It indicates that the collection is a functionally subordinate group of an HID device or a complex device. The operating system uses the Usage associated with this collection to link to the application or driver that controls the device.
0x02	Logical	Used when data items compose a composite data structure.
0x03	Report	Defines a logical collection that includes all fields. A report ID is included in this collection. An application can easily determine whether to support a certain function of the device.
0x04	Named Array	Used when data items compose a composite data structure and it is named.
0x05	Usage Switch	A logical collection that modifies the meaning of the included Usage. It identifies the Usage applied for logical collection to modify the purpose of the Usage being collected.
0x06	Usage Modifier	Modifies the meaning of the Usage attached to the including collection. The Usage typically defines a single operating mode for control, which enables the operating method of control to be expanded.
0x07-7F	Reserved	Reserved.
0x80-FF	Vendor-defined.	Defined by the vendor.

2.6.2 Global Items

Table 2.6 shows 12 item tags for the Global item type.

Table 2.6 Item Tags for Global Item Type

Item Tag	bTag	bType	bSize	Description
Usage Page	0000	01	nn	A value specifying the current Usage Page. It defines the index to the item usage.
Logical Minimum	0001	01	nn	The minimum value to be reported by a variable or array item. For example, the mouse that reports an X position value from 0 to 128 will have a minimum logical value of 0.
Logical Maximum	0010	01	nn	The maximum value to be reported by variable or array items. For example, the mouse that reports an X position value from 0 to 128 will have a maximum logical value of 128.
Physical Minimum	0011	01	nn	Minimum value of physical range for a variable item
Physical Maximum	0100	01	nn	Maximum value of physical range for a variable item
Unit Exponent	0101	01	nn	Unit exponent in base 10
Unit	0110	01	nn	Unit value
Report Size	0101	01	nn	Unsigned value that specifies the report field size in bits
Report ID	1000	01	nn	Unsigned value that specifies the report ID
Report Count	1001	01	nn	Specifies the number of data fields for the item. An unsigned integer specifies how many fields can be included in the report for the particular item (accordingly, how many bits are added to the report).
Push	1010	01	nn	Places a copy of the Global Item state table in the stack
Pop	1011	01	nn	Replaces the item state table with the top data in the stack.

2.6.3 Local Items

Table 2.7 shows ten item tags for the Local item type.

Table 2.7 Item Tags for Local Item Type

Item Tag	bTag	bType	bSize	Description
Usage	0000	10	nn	A value specifying the current Usage. It defines the index to the items usage.
Usage Minimum	0001	10	nn	Defines the start of Usage associated with an array or a bitmap.
Usage Maximum	0010	10	nn	Defines the end of Usage associated with an array or a bitmap.
Designator Index	0011	10	nn	Determines the body part used for control.
Designator Minimum	0100	10	nn	Defines the start index to the designator associated with an array or a bitmap.
Designator Maximum	0101	10	nn	Defines the end index to the designator associated with an array or a bitmap.
String Index	0111	10	nn	Index to the String descriptor, which enables the string to be associated with a particular item or control
String Minimum	1000	10	nn	Specifies the first string index when associating a group of sequential strings to the control in an array or a bitmap.
String Maximum	1001	10	nn	Specifies the end string index when associating a group of sequential strings to the control in an array or a bitmap.
Delimiter	1010	10	nn	Defines the start or end of a set of Local items.

2.6.4 Sample Report Descriptor

Figure 2.3 shows the Report descriptor of this sample program.

```
Usage Page (Generic Desktop),                                       : 05  01
Usage (Mouse),                                                      : 09  02
Collection (Application),                                           : A1  01
    Usage (Pointer),                                                : 09  01
    Collection (Physical),                                          : A1  00
        Usage Page (Buttons),                                       : 05  09
        Usage Minimum (01),                                         : 19  01
        Usage Maximum (03),                                         : 29  03
        Logical Minimum (0),                                        : 15  00
        Logical Maximum (1),                                        : 25  01
        Report Count (3),                                           : 95  03
        Report Size (1),                                            : 75  01
        Input (Data, Variable, Absolute), ; 3 button bits           : 81  02
        Report Count (1),                                           : 95  01
        Report Size (5),                                            : 75  05
        Input (Constant), ; 5 bit padding                           : 81  01
        Usage Page (Generic Desktop),                               : 05  01
        Usage (X),                                                  : 09  30
        Usage (Y),                                                  : 09  31
        Usage (Wheel),                                              : 09  38
        Logical Minimum (-127),                                     : 15  81
        Logical Maximum (127),                                      : 25  7F
        Report Size (8),                                            : 75  08
        Report Count (3),                                           : 95  03
        Input (Data, Variable, Relative), ; 2 position bytes (X & Y) : 81  06
    End Collection,                                                 : C0
End Collection                                                      : C0
```

Figure 2.3 Report Descriptor

2.6.5 Description of Report Descriptor

Table 2.8 shows the Report descriptor used by the sample program.

Table 2.8 Report Descriptor

Item	Value (hex.)	Item Classification	Description
Usage Page (Generic Desktop Control)	0x05 01	Global	A value specifying the Usage Page. 0x01 indicates Generic Desktop Control.
Usage (Mouse)	0x09 02	Local	Index to the item Usage. 0x02 indicates Mouse. The operating system links the device as a mouse to the active application or driver. The Usage type of Mouse is Collection Application.
Collection (Application)	0xA1 01	Main	Notifies the application of Pointer as a mouse.
Usage (Pointer)	0x09 01	Local	Index to the item Usage. 0x01 indicates Pointer. The Usage type of Pointer is Collection Physical.
Collection (Physical)	0xA1 00	Main	Collects multiple sensor positions (button, X axis, Y axis, and rotary control) to one as a pointer.
Usage Page (Button)	0x05 09	Global	A value specifying the Usage Page. 0x09 indicates Button.
Usage Minimum (1)	0x19 01	Local	Defines that the Usage associated with an array or a bitmap starts from 1.
Usage Maximum (3)	0x29 03	Local	Defines that the Usage associated with an array or a bitmap ends at 3.
Logical Minimum (0)	0x15 00	Global	The minimum value to be reported by the item is 0.
Logical Maximum (1)	0x25 01	Global	The maximum value to be reported by the item is 1.
Report Count (3)	0x95 03	Global	Indicates the number of data fields to be used for the item. This example indicates that three report fields are to be used.
Report Size (1)	0x75 01	Global	Indicates the report field size. This example indicates that 1-bit field is to be used.
Input (Data, Variable, Absolute)	0x81 02	Main	Indicates the type of input item. This example indicates that the input is variable data and reports an absolute value.

Item	Value (hex.)	Item Classification	Description
Report Count (1)	0x95 01	Global	Indicates the number of data fields to be used for the item. This example indicates that one report field is to be used.
Report Size (5)	0x75 05	Global	Indicates the report field size. This example indicates that 5-bit field is to be used.
Input (Constant)	0x81 01	Main	Indicates the type of input item. This example indicates that the input reports a constant.
Usage Page (Generic Desktop Control)	0x05 01	Global	A value specifying the Usage Page. 0x01 indicates Generic Desktop Control.
Usage (X)	0x09 30	Local	Index to the item Usage. 0x30 indicates X. The controller reports X-direction values, and when the controller moves from left to right from the user's viewpoint, a value increases linearly.
Usage (Y)	0x09 31	Local	Index to the item Usage. 0x31 indicates Y. The controller reports Y-direction values, and when the controller moves from the far side to the near side from the user's viewpoint, a value increases linearly.
Usage (Wheel)	0x09 38	Local	Index to the item Usage. 0x38 indicates Wheel. It is different from a dial; it is a rotary control that generates a variable value when rotated. When the controller rotates toward the front (the far side from the user), a value increases.
Logical Minimum (-127)	0x15 81	Global	The minimum value to be reported by the item is -127.
Logical Maximum (127)	0x25 7F	Global	The maximum value to be reported by the item is 127.
Report Size (8)	0x75 08	Global	Indicates the report field size. This example indicates that 8-bit field is to be used.
Report Count (3)	0x95 03	Global	Indicates the number of data fields to be used for the item. This example indicates that three report fields are to be used.
Input (Data, Variable, Relative)	0x81 06	Main	Indicates the type of input item. This example indicates that the input is variable data and reports the change from the last input.
End Collection	0xC0	Main	Indicates the end of collection of data set (physical).
End Collection	0xC0	Main	Indicates the end of collection of data set (application).

2.7 Physical Descriptor

The physical descriptor provides information about the human body (or a specific part of the human body) that is controlling the device. This descriptor is optional, and it is omitted in the sample program.

2.8 HID Data Transfer Format

HID data is transferred between the host PC and the USB function module mainly through interrupt transfers (control transfers are also available).

The boot device can use two types of protocols: report protocol and boot protocol. Other devices can only use one protocol: report protocol.

The format of data transfer used by the report protocol is described by a Report descriptor. The format used by the boot protocol is prescribed in the USB standard.

The default protocol for the boot device is the report protocol, but a class command can select either the boot or report protocol. Figure 2.4 shows the report protocol format used by the sample program.

Bits:	7 to 3	2	1	0			
Parts:	00000	Wheel button	Right button	Left button	X axis	Y axis	Wheel
Byte:		0			1	2	3

Figure 2.4 Report Protocol Format

2.9 Class Commands

Class commands are defined by each USB class. They use control transfer.

There are six commands for the USB HID class. Table 2.9 shows the class commands.

Table 2.9 Class Commands

bRequest Field Value	Command	Meaning of Command
0x01	GET_REPORT	Transfers HID data from the device to the host PC through control transfer
0x02	GET_IDLE	Returns the current value for the rate of time for which interrupt transfer stops
0x03	GET_PROTOCOL	Reports the current active protocol (boot protocol or report protocol)
0x09	SET_REPORT	Transfers HID data from the host PC to the device through control transfer
0x0A	SET_IDLE	Specifies the rate of time for which interrupt transfer stops
0x0B	SET_PROTOCOL	Specifies the active protocol (boot protocol or report protocol)

Notes: 1. All devices must support GET_REPORT.
2. Boot devices must support GET_PROTOCOL and SET_PROTOCOL.

When the GET_REPORT command is received, the function sends HID data to the host through the data stage of control transfer. The report type must be specified in the upper one byte of the wValue field in the setup data and the report ID in the lower one byte of the wValue field.

When the GET_IDLE command is received, the function returns the time for which interrupt transfer stops. The time should be expressed in time rate in 4-ms units. The host specifies the ID for the report that the host requests in the lower one byte of the wValue field in the setup data. If this value is 0, the time rates for all interrupt transfers of the target device are returned.

When the GET_PROTOCOL command is received, the function returns the current active protocol (boot protocol or report protocol) to the host through the data state of control transfer. Value 0 indicates the boot protocol, and value 1 indicates the report protocol.

When the SET_REPORT command is received, the function receives HID data through the data stage of control transfer. However, the function may ignore the command from the host.

When the SET_IDLE command is received, the function stops interrupt transfer for the time specified in the upper one byte of the wValue field in the setup data. The time is expressed in time rate in 4-ms units. The lower one byte of the wValue field specifies the report ID. If this value is not 0, the transfer of the specified report ID is stopped. If this value is 0, all interrupt transfers of the target device are stopped.

When the SET_PROTOCOL command is received, the function specifies the protocol (boot protocol or report protocol) to be used from that time on. The protocol is specified in the wValue filed in the setup data (value 0 indicates the boot protocol and value 1 indicates the report protocol). Note that the report protocol is the default protocol of the function.

Section 3 Development Environment

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

- H8S/2218 Solution Engine (hereafter called the MS2218CP; type number: MS2218CP01) manufactured by Hitachi ULSI Systems Co., Ltd.
- E10A Emulator manufactured by Renesas Technology Corp.
- PC (Windows® 95/Windows®98/Windows®Me/Windows® 2000/Windows® XP) equipped with a PCMCIA, PCI, ISA, or USB slot
- PC (Windows®Me/Windows® 2000/Windows® XP) to serve as the USB host
- USB cable
- Debugging Interface (hereafter called the HDI) manufactured by Renesas Technology Corp.
- High-Performance Embedded Workshop (hereafter called the HEW) manufactured by Renesas Technology Corp.

3.1 Hardware Environment

Figure 3.1 shows device connections.

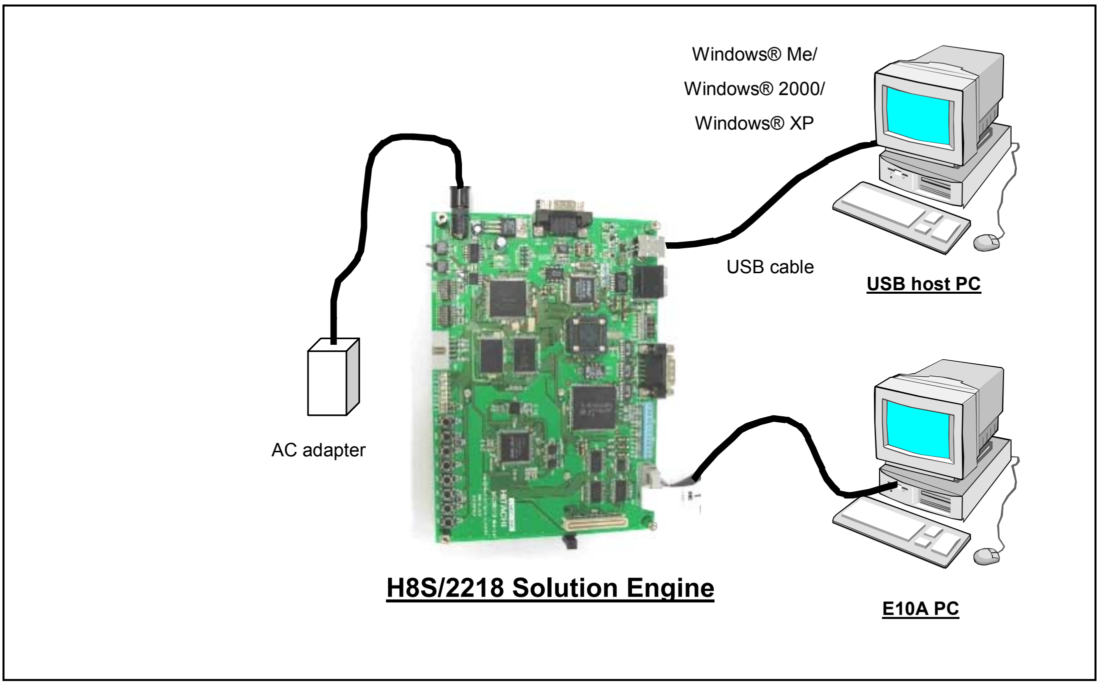


Figure 3.1 Device Connections

1. MS2218CP

   Some DIP switch and jumper settings on the MS2218CP board must be changed from those at shipment. Before turning on the power, ensure that the switches and jumpers are set as follows. There is no need to change any other switches and jumpers.

Table 3.1 Switch and Jumper Settings

At Shipment	After Change	Function
SW1-1 Off	SW1-1 On	Selects operation mode 6
SW1-2 Off	SW1-1 Off	
SW1-3 Off	SW1-1 Off	
SW1-5 Off	SW1-1 On	Selects the E10A emulator mode
J-3 Closed	J-3 Open	Selects the USB self-powered mode
J-9 Closed	J-9 Open	Selects the big endian mode.

2. USB host PC

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

3. E10A

   The PCMCIA is used for the communication interface between the E10A PC and the E10A emulator.

   The E10A card should be inserted into a PC card slot and connected to the MS2218CP via an interface 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 H8S2218 folder. When this entire folder with its contents is moved to a PC on which HEW and HDI have been installed, the sample program can be used immediately. Files included in the folder are shown in figure 3.2 below.

H8S2218

CatHidTypedef.h	CatProTypedef.h	CatTypedef.h	h8s2218.h	
SetHidInfo.h	SetMacro.h	SetSystemSwich.h	SetUsbInfo.h	
SysMemMap.h				
DoControl.c	DoHidDataFormat.c	DoInterrupt.c	DoMouse.c	
DoRequest.c	StartUp.c	UsbMain.c	DoRequest HIDClass.c	
ch38iop (folder)	dwfinf (folder)	log.txt	ReadMe.txt	InkSet1.sub
sct.src		debugger.MAP	debugger.HDW	
debugger.HDT	debugger.MOT	debugger.ABS	BuildOfHew.bat	
	debugger.hds			

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.

High-Performance 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).

```
C:\
  └── \Hew
        └── \Tmp
```

Figure 3.3 Creating a Working Folder

Next, the folder in which the sample program is stored (H8S2218) should be copied to C:\Usr (or can be copied to any location, then "C:\Usr\h8s2218" 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 BuildOfHew.bat. This batch file sets the path, specifies compile options, specifies a log file indicating the compile and linking results, and performs other operations. When BuildOfHew.bat is executed, compiling and linking are performed. As a result, an executable file named debugger.MOT, which is a file in the Motorola S-type format, is created within the folder. At the same time, a map file named debugger.MAP and a log file named log.txt are created. The map file indicates the program size and addresses of variables. The compile results (whether there are any errors etc.) are recorded in the log file. (Figure 3.4)

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

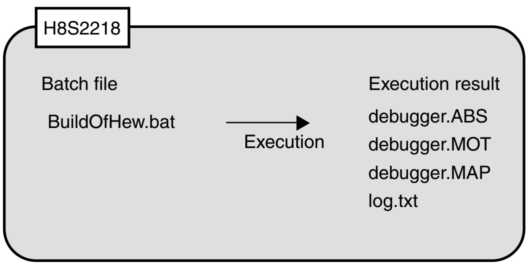
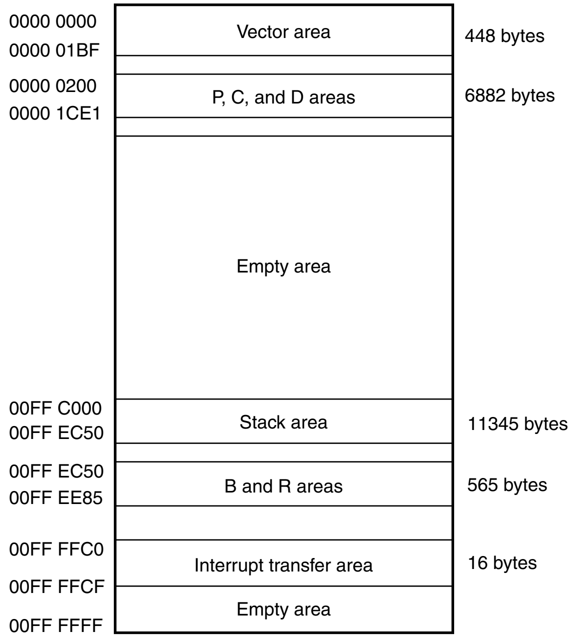


Figure 3.4 Compile Results

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 flash memory area in area 0, and the stack, B, and R areas to the on-chip RAM. These memory allocations are specified by the InkSet1.sub file in the H8S2218 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 MS2218CP, the following procedure is used.

- Insert the E10A into the E10A PC in which the HDI has been installed.
- Connect the E10A to the MS2218CP via an E10A cable.
- Turn on the power to the MS2218CP to start up the machine.
- Execute debugger.hds in the H8S2218 folder.
- When the operating frequency is asked, enter the frequency of the installed crystal resonator (16 or 24 MHz)
- When the registry is asked, enter 0.

Through the above procedure, the E10A starts operation.

3.3.2 Executing the Program

In order to execute the program which was loaded in section 3.3.1 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 write the program to the on-chip flash memory and execute it.

3.4 Demonstrating Pseudo Mouse Operation (Cursor Movements)

The sample program demonstrates pseudo mouse operation (cursor movements) without a mouse connected.

While the program is running, connect series-B connector of the USB cable to the MS2218CP, and series-A connector to the USB host PC. After control transfer is completed, the human interface devices and USB human interface devices are displayed in the device manager window, and the host PC recognizes the MS2218CP as a mouse device.

After the MS2218CP is connected to the host PC, pressing switches SW5 to SW12 generates data of mouse movements.

Pressing SW5 generates data for the left mouse button pressed, and SW6 for the right mouse button pressed. Pressing SW8 demonstrates pointer movements. Pressing SW9 generates data for pointer movement downward in the Y direction, SW10 for pointer movement upward in the Y direction, SW11 for pointer movement toward right in the X direction, and SW12 for pointer movement toward left in the X direction. The MS2218CP outputs pseudo mouse data in response to interrupt-in transfer from the host PC. As a result, the cursor on the USB host PC automatically starts moving.

Section 4 Overview of the Sample Program

In this section, features of the sample program and its structure are explained. This sample program is an HID class firmware, which runs on the MS2218CP and generates mouse data to emulate mouse movements. The sample program initiates USB transfers by means of tokens from the host PC. Of the interrupts from modules in the H8S/2218, 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.
- Interrupt-in transfer can be used to send pseudo mouse data to the host PC.

4.1 State Transition Diagram

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

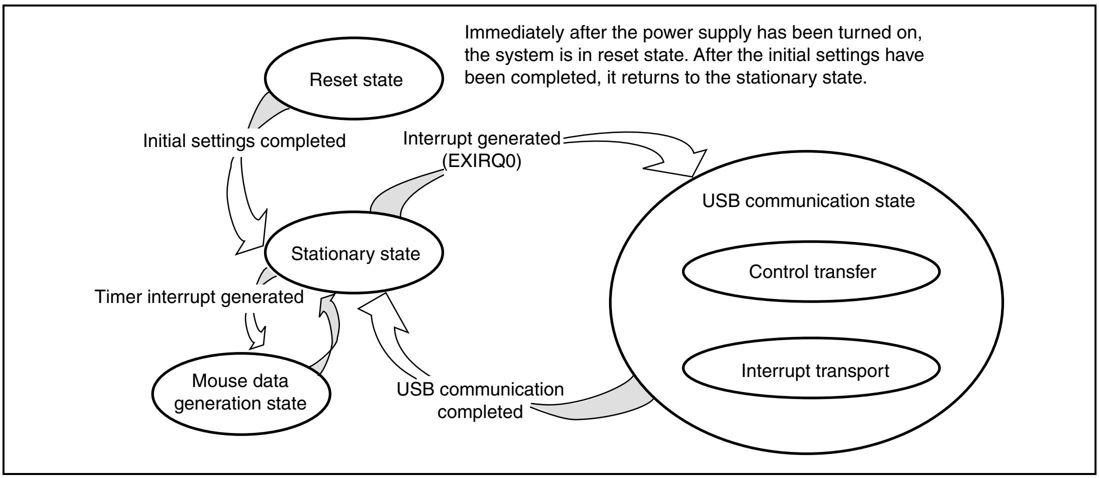


Figure 4.1 State Transition Diagram

- Reset State

  Upon power-on reset and manual reset, this state is entered. In the reset state, the H8S/2218 mainly performs initial settings.
- Stationary State

  When initial settings are completed, a stationary state is entered in the main loop.

- USB Communication State

  In the stationary state, when an interrupt from the USB module occurs, this state is entered. In the USB communication state, data transfer is performed by a transfer method according to the type of interrupt. The interrupts used in this sample program are indicated by interrupt flag registers 0, 1, and 3 (UIFR0, UIFR1, and UIFR3), and there are nine interrupt types in all. When an interrupt factor occurs, the corresponding bits in UIFR0, UIFR1, or UIFR3 are set to 1.

- Mouse Data Generation State

  In the stationary state, when a compare match interrupt from 16-bit timer TGRA_2 occurs, this state is entered. In the mouse data generation state, mouse data is generated by pressing switches on the board or is automatically generated without a mouse connected. A compare match interrupt occurs every 16 ms or 10 ms.

4.2 USB Communication State

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

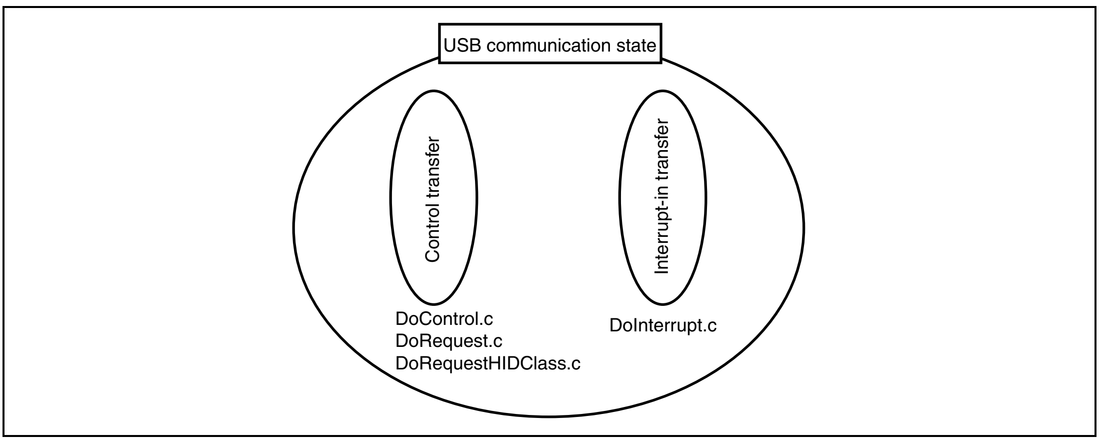


Figure 4.2 USB Communication State

4.3 File Structure

This sample program consists of eight 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

File Name	Principle Role
StartUp.c	Microcomputer default settings
UsbMain.c	Judging the causes of interrupts Sending and receiving packets
DoControl.c	Executing control transfer
DoInterrupt.c	Executing interrupt-in transfer
DoRequest.c	Processing setup commands issued by the host
DoRequestHIDClass.c	Processing HID class commands
DoHidDataFormat.c	Formatting HID data to be transferred
DoMouse.c	Generating mouse data
CatHidTypedef.h	Defining types and structures specific to HID class
CatProType.h	Declaring prototypes
CatTypedef.h	Defining the basic structures used in USB firmware
h8s2218.h	Defining H8S/2218 registers
SetHidInfo.h	Default settings of variables needed to support HID class
SetMacro.h	Defining macros
SetSystemSwitch.h	System operation settings
SetUsbInfo.h	Default settings of variables needed to support USB firmware
SysMemMap.h	Defining MS2218CP memory map addresses

4.4 Purposes of Functions

Tables 4.2 to 4.9 show functions contained in each file and their purposes.

Table 4.2 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 supported, not used by this sample program)
	GetPacket4S	Writes data transferred from the host controller to RAM in longwords (ring buffer not supported, high-speed version)
	PutPacket	Writes data for transfer to the host controller to the USB module
	PutPacket4	Writes data for transfer to the host controller to the USB module in longwords (ring buffer supported, not used by this sample program)
	PutPacket4S	Writes data for transfer to the host controller to the USB module in longwords (ring buffer not supported, high-speed version)
	SetControlOutContents	Overwrites data with that sent from the host
	SetUsbModule	Makes USB module initial settings
	ActBusReset	Clears FIFO on receiving bus reset
	ActBusVcc	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 registers, and functions are called according to the interrupt type. Also, packets are sent and received between the host controller and function modules.

Table 4.3 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 that have default settings to the RAM work area
	InitMemory	Allocates memory areas
	InitSystem	Specifies the USB clock, system interrupt masks, and timers

When a power-on reset or manual reset is carried out, the SetPowerOnSection of the StartUp.c file is called. At this point, initial settings for the H8S/2218 registers or USB clock are performed.

Table 4.4 DoRequest.c

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

During control transfer, commands sent from the host controller are decoded and processed. In this sample program, a vendor ID of 045B 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 4.5 DoRequestHIDClass.c

File in Which Stored	Function Name	Purpose
DoRequestHIDClass.c	DecHIDClassCommands	Processes HID class commands
	ActIdleCount	This is called by an SOF interrupt, and counts the time for which interrupt transfer stops

These functions carry out processing according to the HID class commands (GET_REPORT, GET_IDLE, GET_PROTOCOL, SET_REPORT, SET_IDLE, and SET_PROTOCOL).

The GET_REPORT command sends HID data from the device to the host PC through control transfer.
The GET_IDLE command returns the rate for the time for which interrupt transfer stops.
The GET_PROTOCOL command returns the current active protocol (boot protocol or report protocol).

The SET_REPORT command sends HID data from the host PC to the device through control transfer, but this sample program does not support out-direction communications of HID data and only receives data.
The SET_IDLE command specifies the rate for the time for which interrupt transfer stops.
The SET_PROTOCOL command specifies the active protocol (boot protocol or report protocol).

Table 4.6 DoControl.c

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

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

Table 4.7 DoInterrupt.c

File in Which Stored	Function Name	Purpose
DoInterrupt.c	ActInterruptIn	On receiving the in-token of the interrupt transfer, gets data from the data transfer buffer as soon as FIFO has an empty space and prepares for interrupt transfer

On receiving the in-token of the interrupt transfer from the host PC, this function prepares next data to be sent as soon as the interrupt transfer buffer becomes empty.

Table 4.8 DoHidDataFormat.c

File in Which Stored	Function Name	Purpose
DoHidDataFormat.c	ActMakeHidData	A program interface for HID data communications. Calls ActInterruptIn if interrupt transfer stops after ActReportProtocol is called.
	ActReportProtocol	Arranges transfer data according to the format specified by the Report descriptor, and writes the data to the transmit buffer.

These functions prepare HID data to be transmitted to the host PC.

Table 4.9 DoMouse.c

File in Which Stored	Function Name	Purpose
DoMouse.c	MousePushedDataInput	This is initiated by a timer interrupt, and determines whether to perform key scan or to generate data for mouse movements.
	MousePushedDataInput1	Performs key scan and generates mouse data.
	MousePushedDataInput2	Generates data for mouse movements according to the time counted by the timer.

DoMouse.c uses a timer interrupt and generates mouse data.

Figure 4.3 shows the interrelationship between the functions explained in tables 4.2 to 4.9. The upper-side functions can call the lower-side functions. Also, multiple functions can call the same function. In the stationary state, SetPowerOnSection calls other functions, and in the 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.

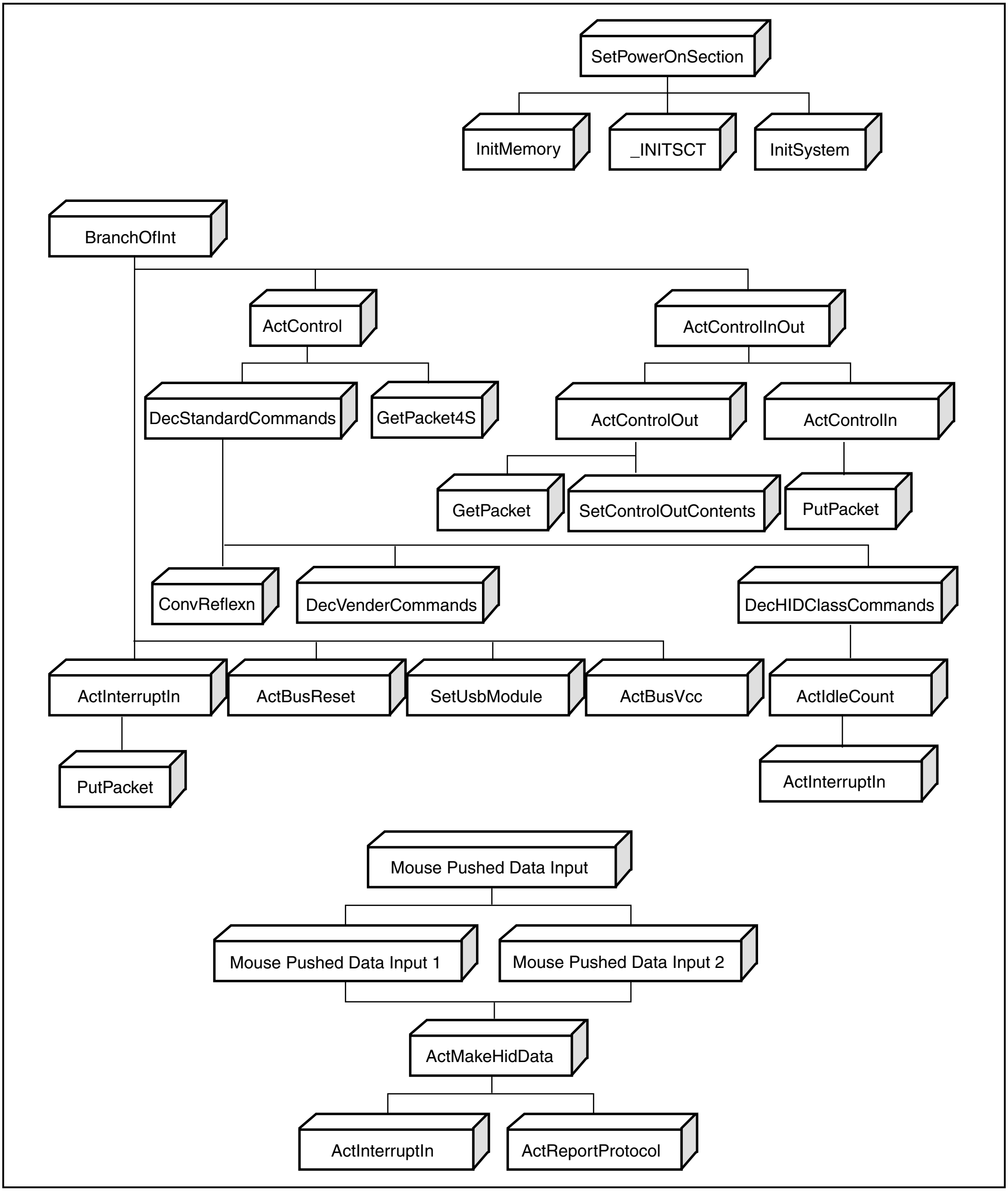


Figure 4.3 Interrelationship between Functions

Section 5 Sample Program Operation

In this section, 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 on-chip peripheral modules are initialized. Next, the function SetPowerOnSection in StartUp.c is called, and the CPU is initialized. Figure 5.1 is a flow chart for the SetPowerOnSection function operation.

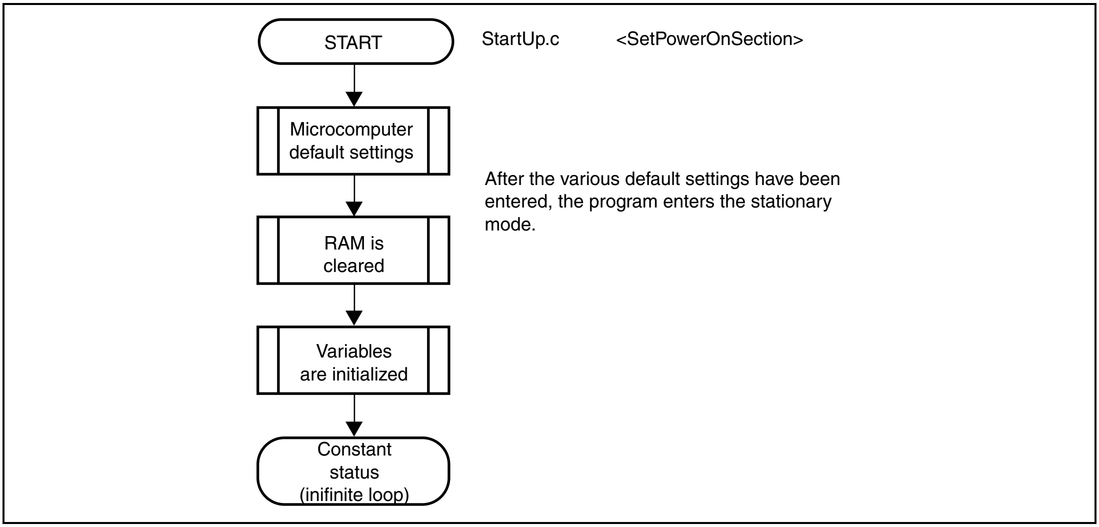


Figure 5.1 Main Loop

5.2 Types of Interrupts

As explained in section 4, the interrupts used in this sample program are indicated by the interrupt flag registers 0, 1, and 3 (UIFR0, UIFR1, and UIFR3); there are a total of nine types of interrupts. When an interrupt factor occurs, the corresponding bits in the interrupt flag registers 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.

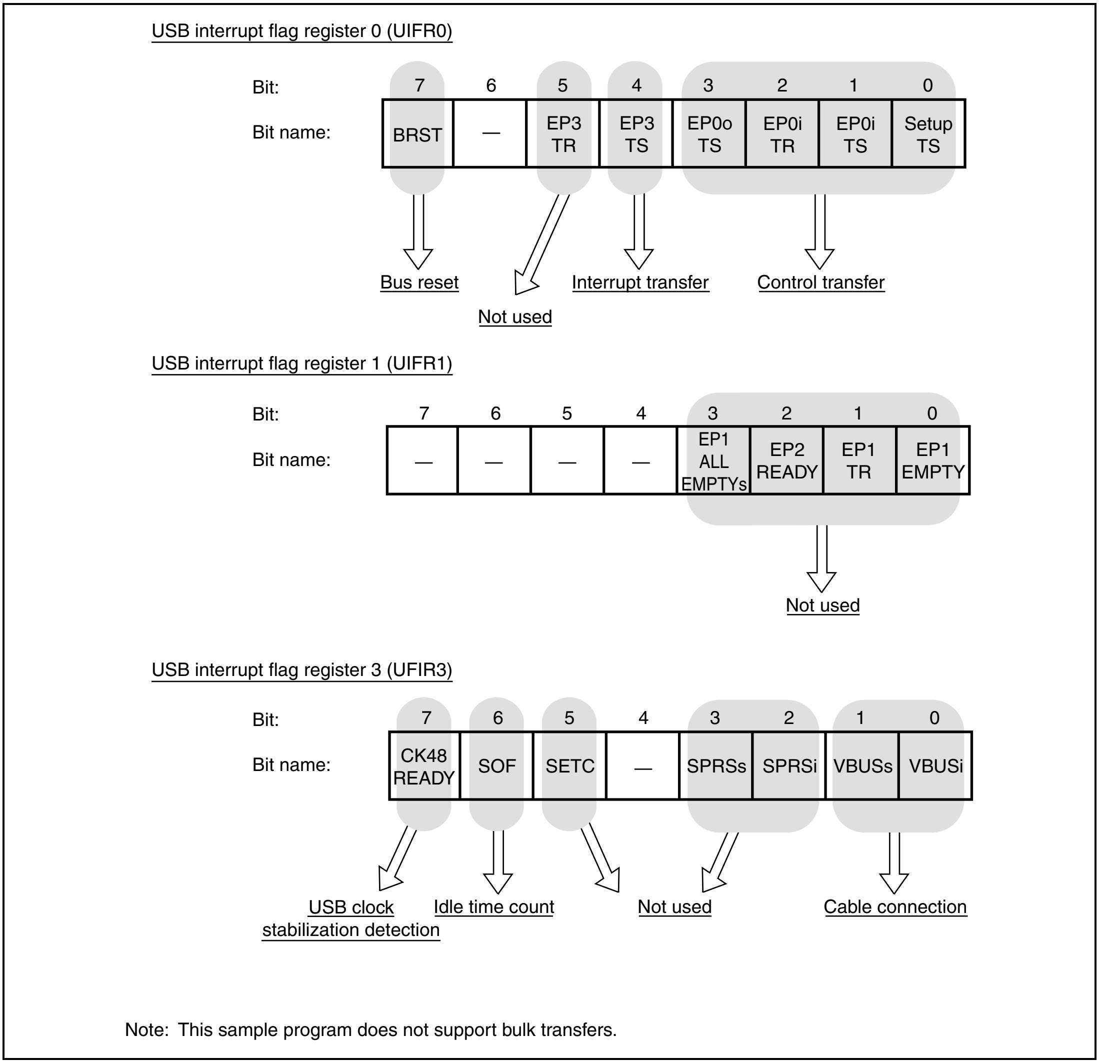


Figure 5.2 Types of Interrupt Flags

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. Branching to each transfer method 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
UIFR0	7	BRST	ActBusReset
	6	—	—
	5	EP3 TR	—
	4	EP3 TS	ActInterruptIn
	3	EP0o TS	ActControlInOut
	2	EP0i TR	ActControlInOut
	1	EP0i TS	ActControlInOut
	0	SETUP TS	ActControl
UIFR3	7	CK48 READY	SetUSBModule
	6	SOF	ActIdleCount
	5	SETC	—
	4	—	—
	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/2218 Group, 2212 Group 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 waits for USB cable connection.

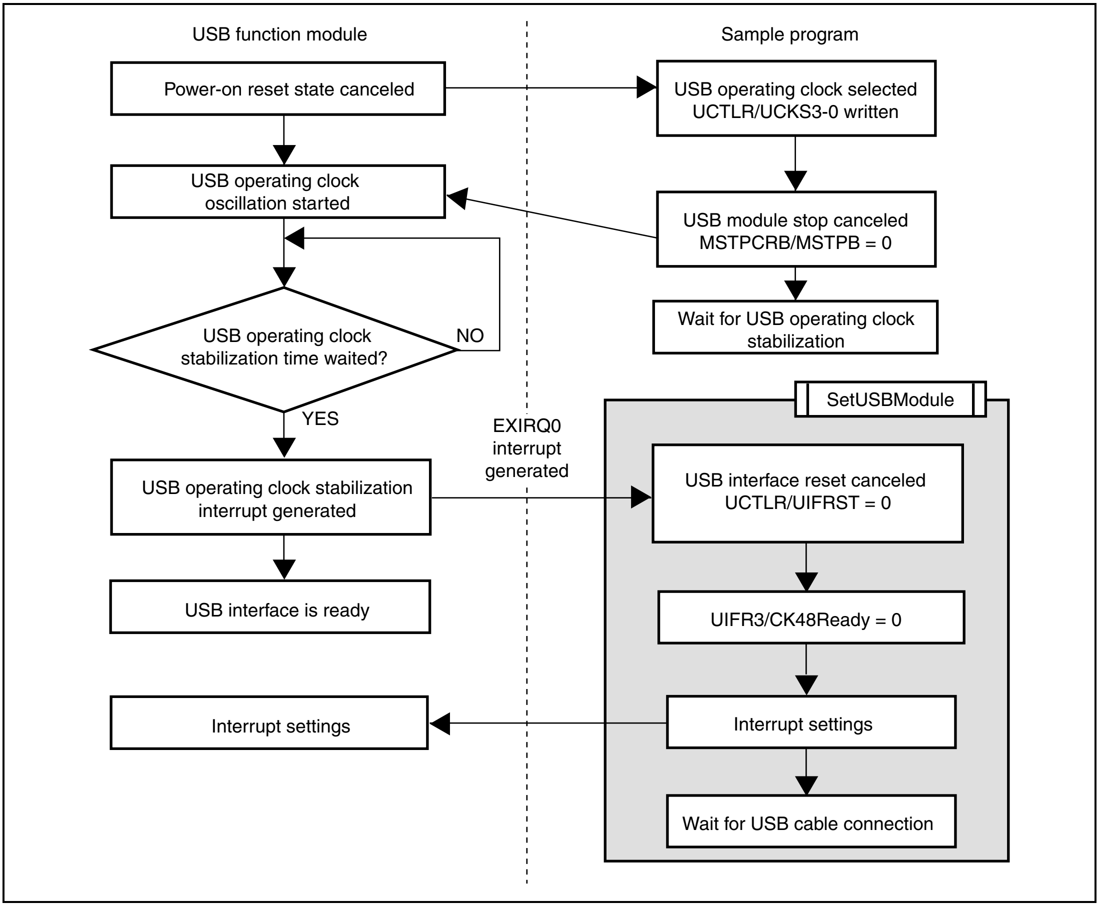


Figure 5.3 USB Operating Clock Stabilization Interrupt

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 5.4)

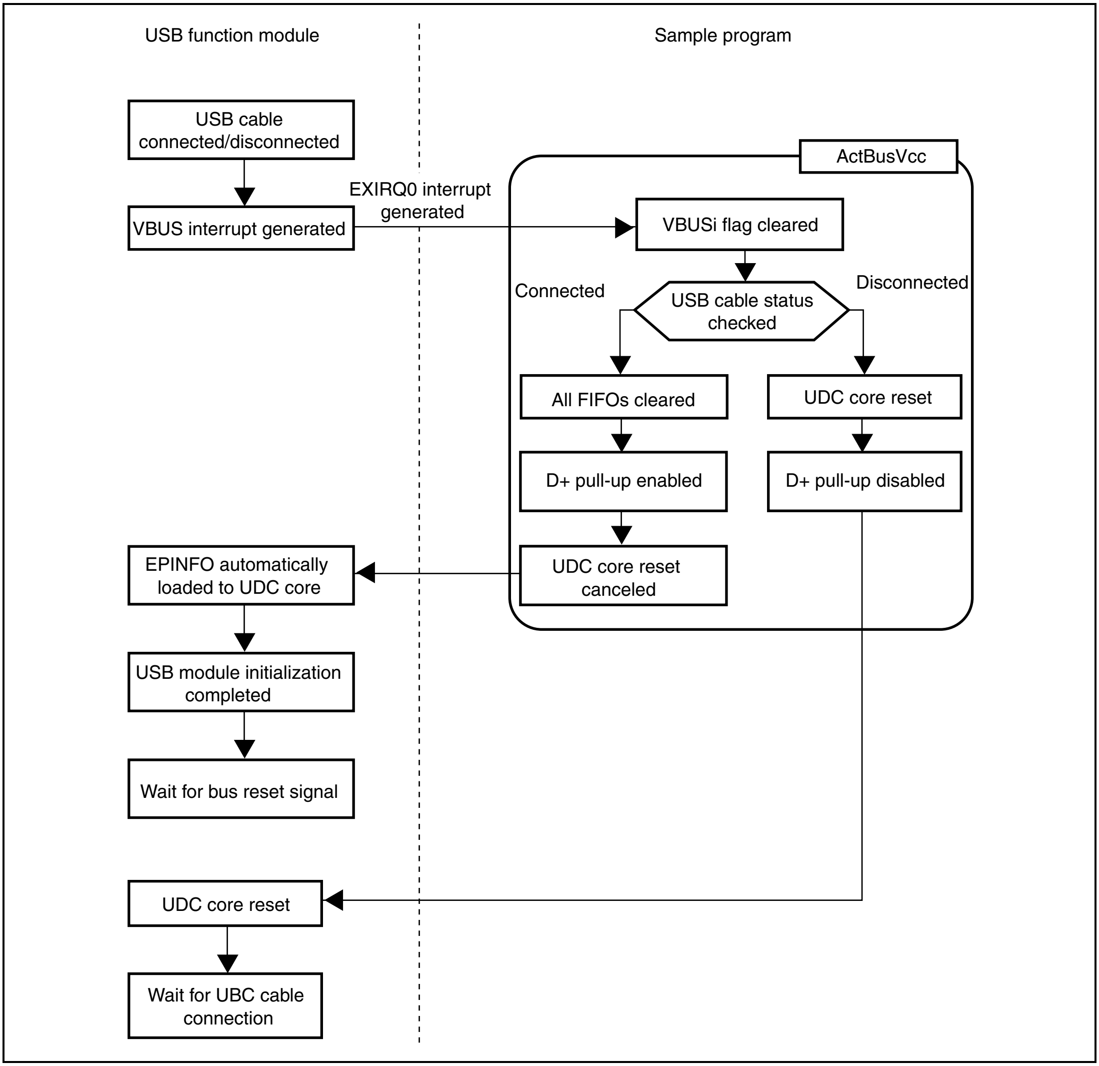


Figure 5.4 Interrupt on Cable Connection

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 interrupt.

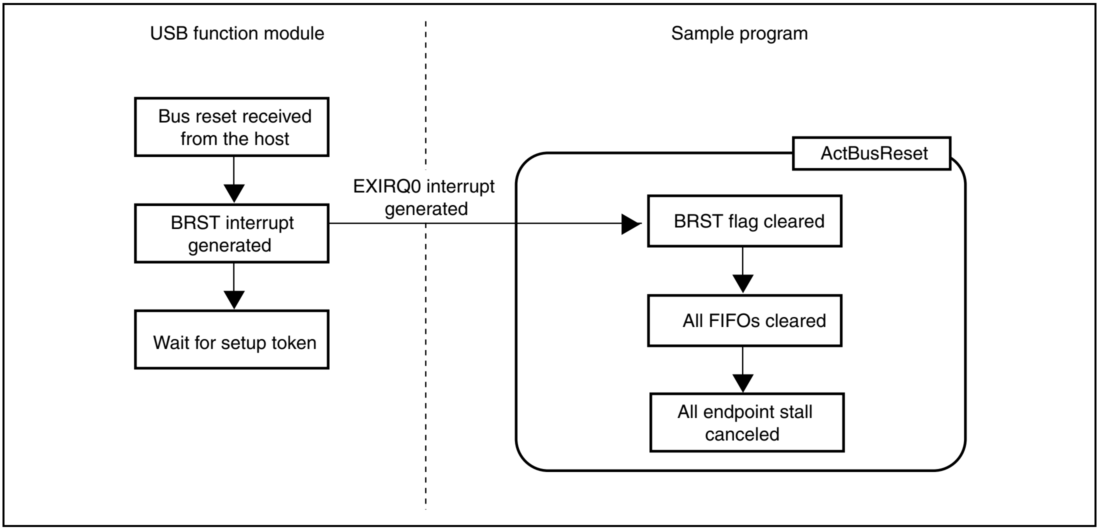


Figure 5.5 Bus Reset Interrupt

5.6 Control Transfers

In control transfers, bits 0, 1, and 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 5.6). 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.

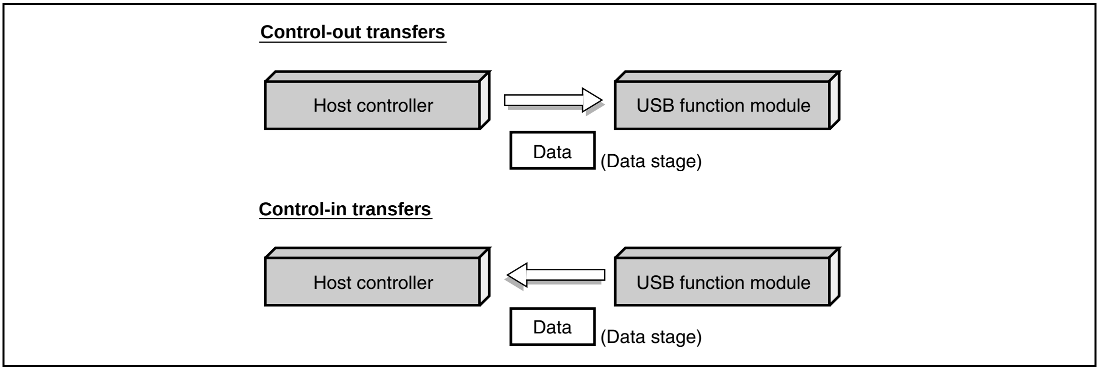


Figure 5.6 Control Transfers

Control transfers consist of three stages: setup, data (no data is possible), and status (figure 5.7). Further, the data stage consists of multiple bus transactions.

In control transfers, stage changes are recognized through the reversal of the data direction. Hence the same interrupt flag is used to call a function to perform control-in or control-out transfers (table 5.1). For this reason, the firmware must use states to manage the type of control transfer currently being performed, whether control-in or control-out (figure 5.7), 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.

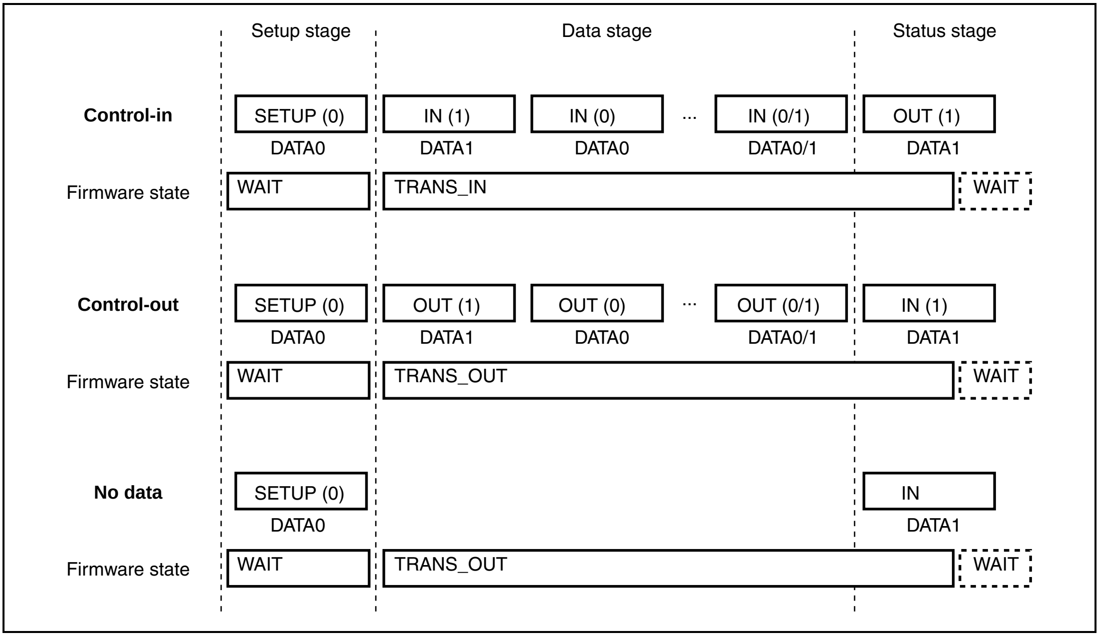


Figure 5.7 Status in Control Transfers

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.

- Command for control-in transfers: GetDescriptor (Standard command)

Figure 5.8 shows operation of the sample program in the setup stage. The figure on the left shows operation of the USB function module.

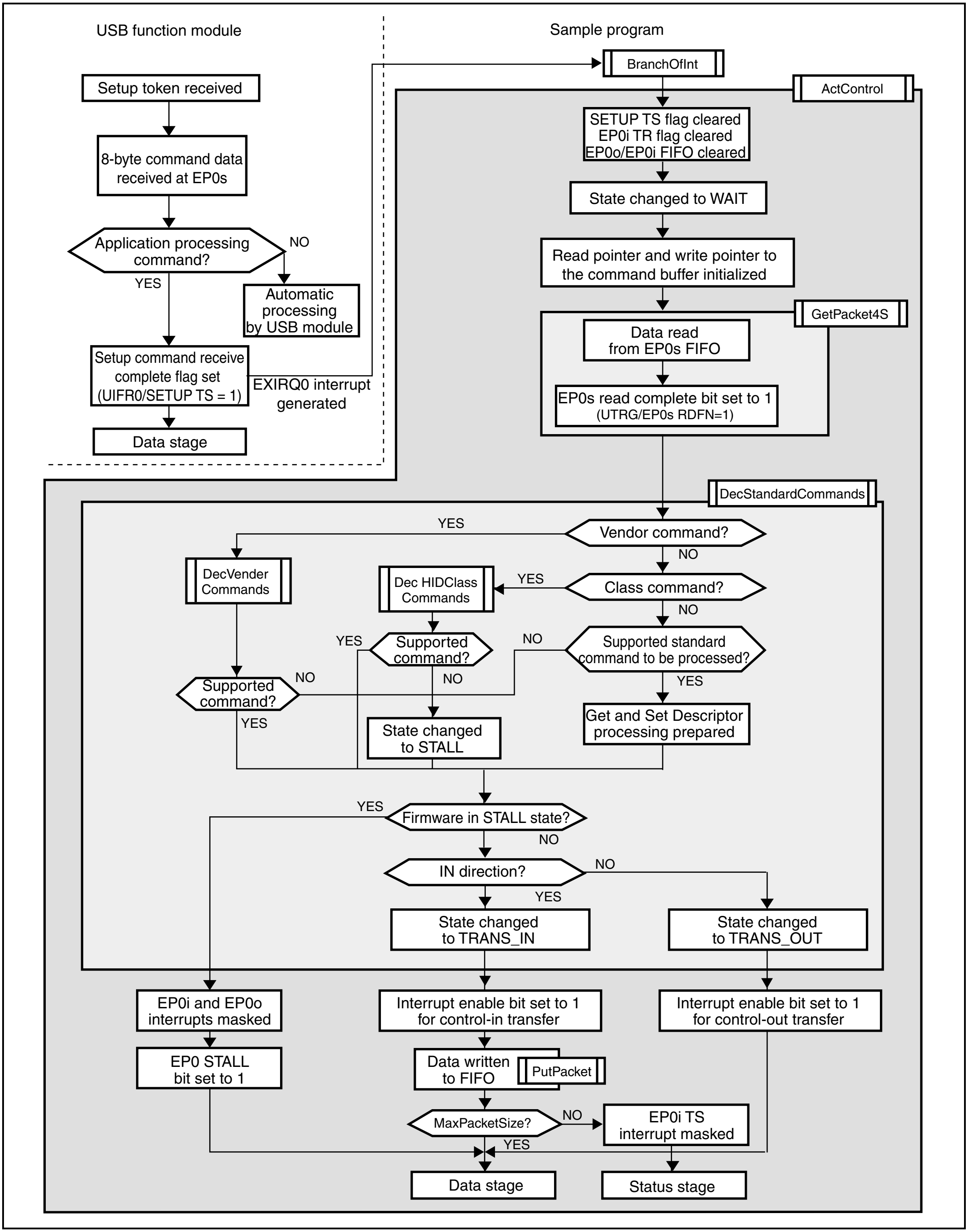


Figure 5.8 Setup Stage

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 5.9 and 5.10 show the operation of the sample program in the data stage of control transfer.

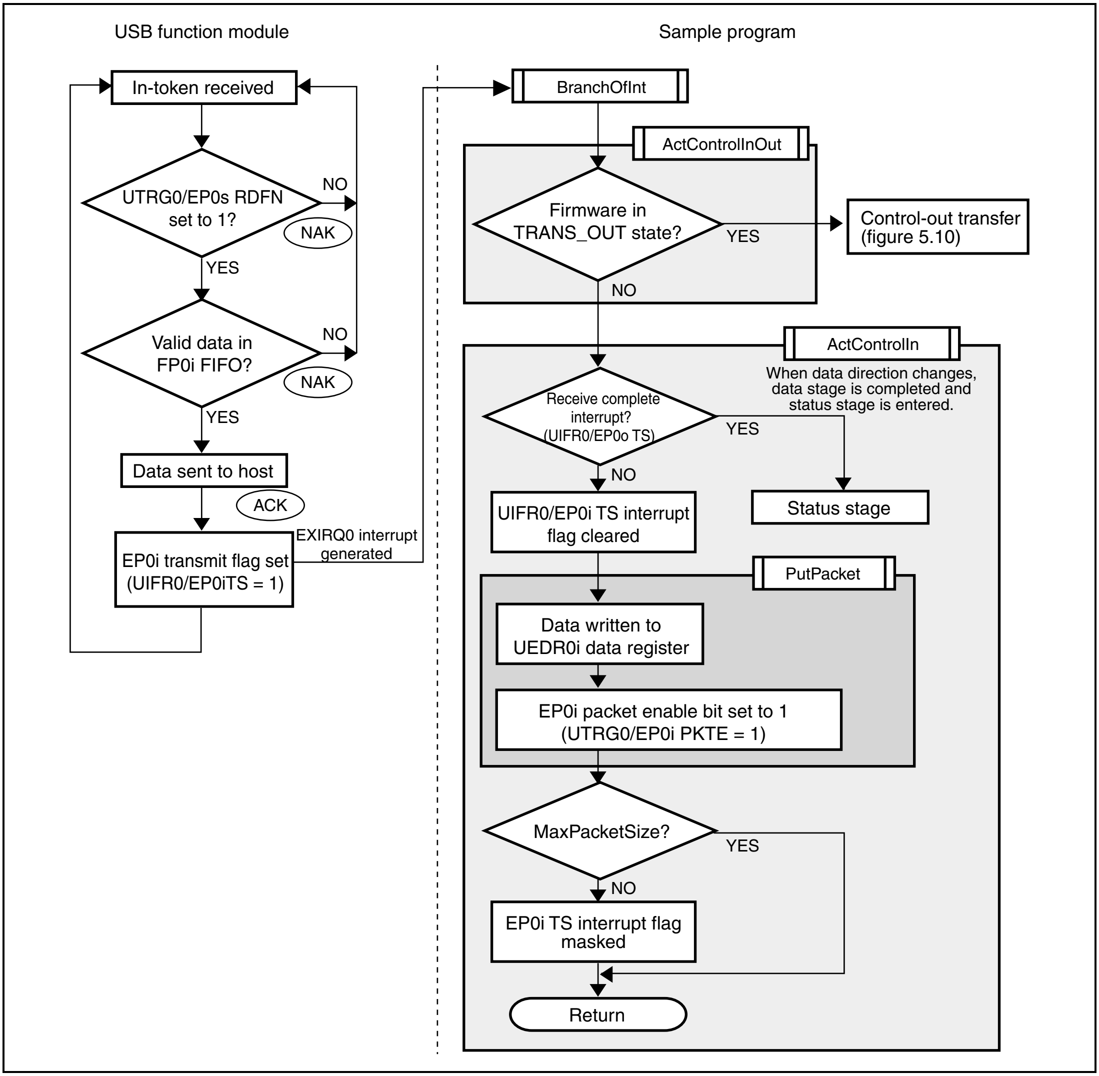


Figure 5.9 Data Stage (Control-In Transfer)

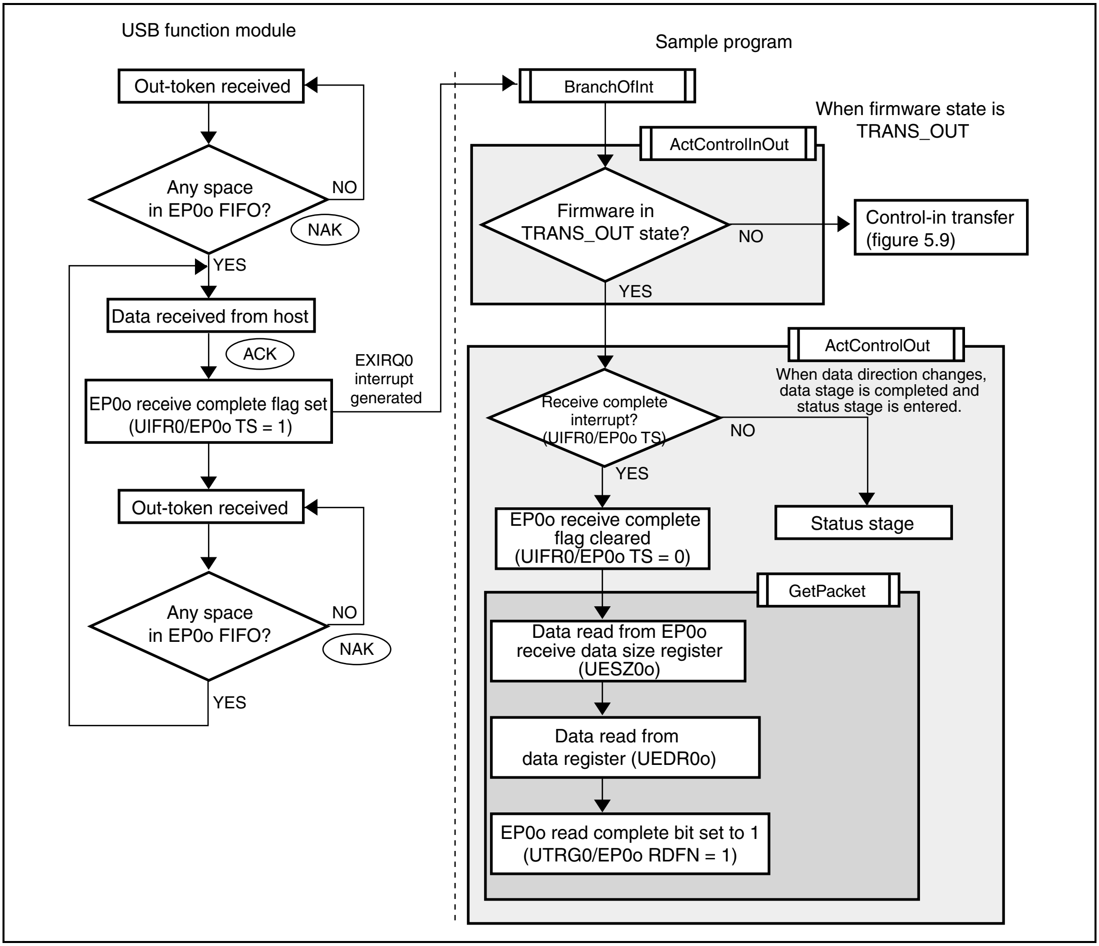


Figure 5.10 Data Stage (Control-Out Transfer)

RENESAS

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. Figures 5.11 and 5.12 show the operation of the sample program in the status stage of control transfer.

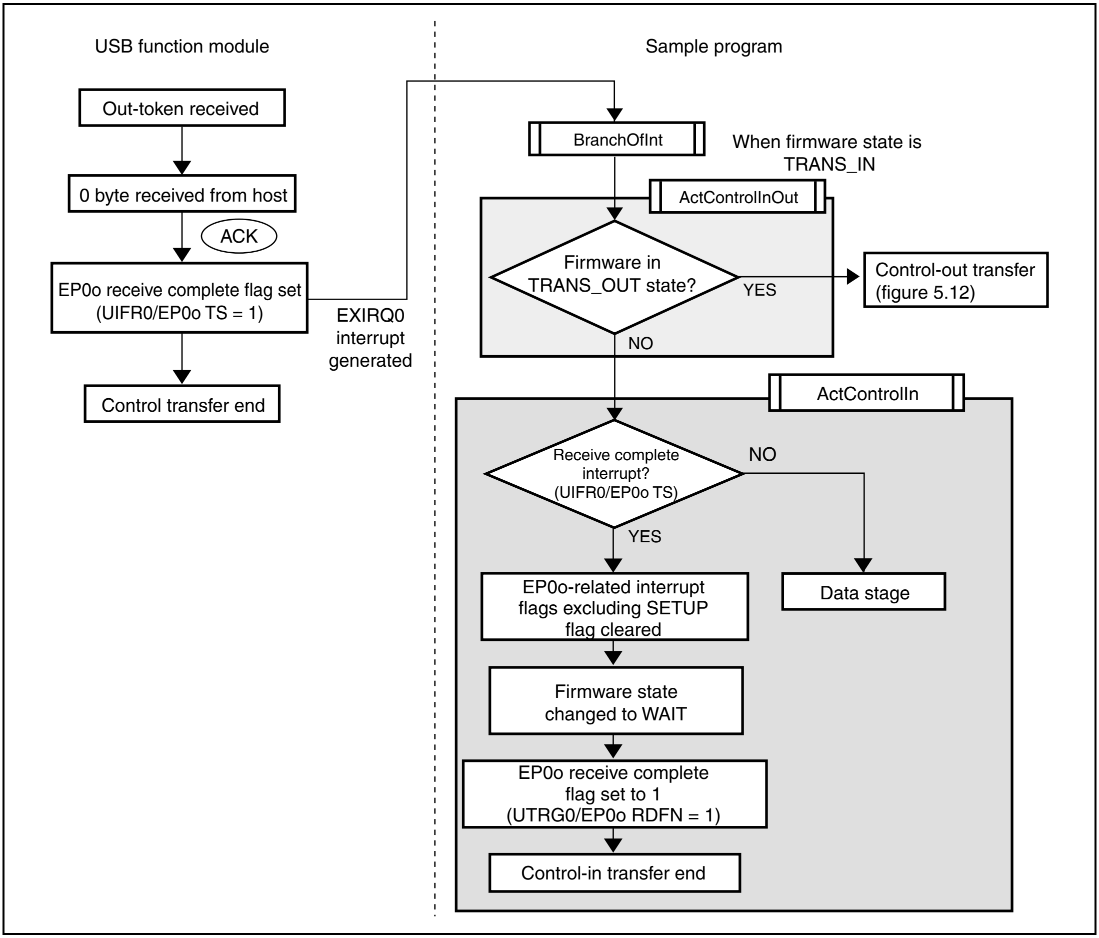


Figure 5.11 Status Stage (Control-In Transfer)

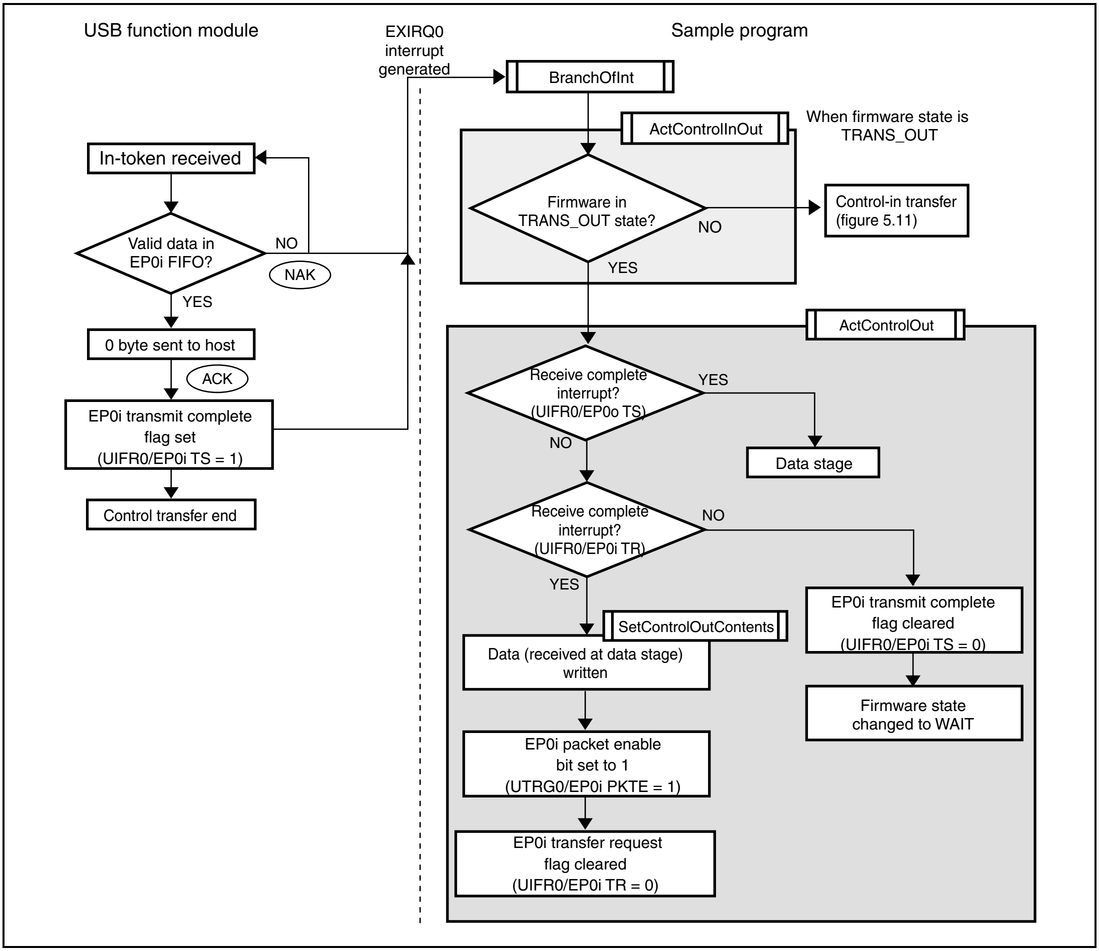


Figure 5.12 Status Stage (Control-Out Transfer)

RENESAS

5.7 Interrupt Transfers

Interrupt transfers can also be classified into two types according to the direction of data transmission. Data transfers from the USB function module to the host controller are interrupt-in transfers, and transfers in the opposite direction are interrupt-out transfers. The H8S/2218 only supports interrupt-in transfers (figure 5.13).

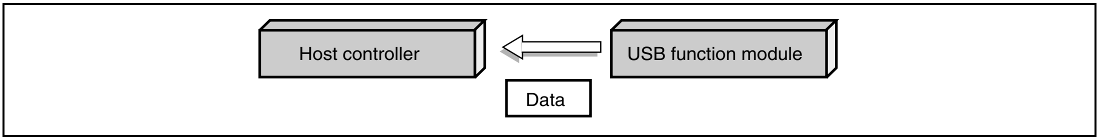


Figure 5.13 Interrupt Transfers

5.7.1 Interrupt-In Transfers

In interrupt-in transfers, bit 4 (EP3TS) of interrupt flag register 0 is used. On receiving an in-token from the USB host controller, the USB function module sends the NAK handshake and sets the EP3TR flag if no valid data is found in the EP3 FIFO. If valid data is found in the FIFO, the USB function module sends data to the USB host controller, and sets the EP3TS flag when receiving the ACK handshake from the USB host controller.

After the EP3TS flag is set, the USB function module executes the ActInterruptIn function. When there is HID data to be sent, this function writes the data to USB endpoint data register 3 (UEDR3) and waits for an in-token to be sent from the USB host controller. At this point, the firmware is in either WAIT or TRANS_IN state. Figure 5.14 shows operation of the sample program in interrupt-in transfer. The figure on the left shows operation of the USB function module.

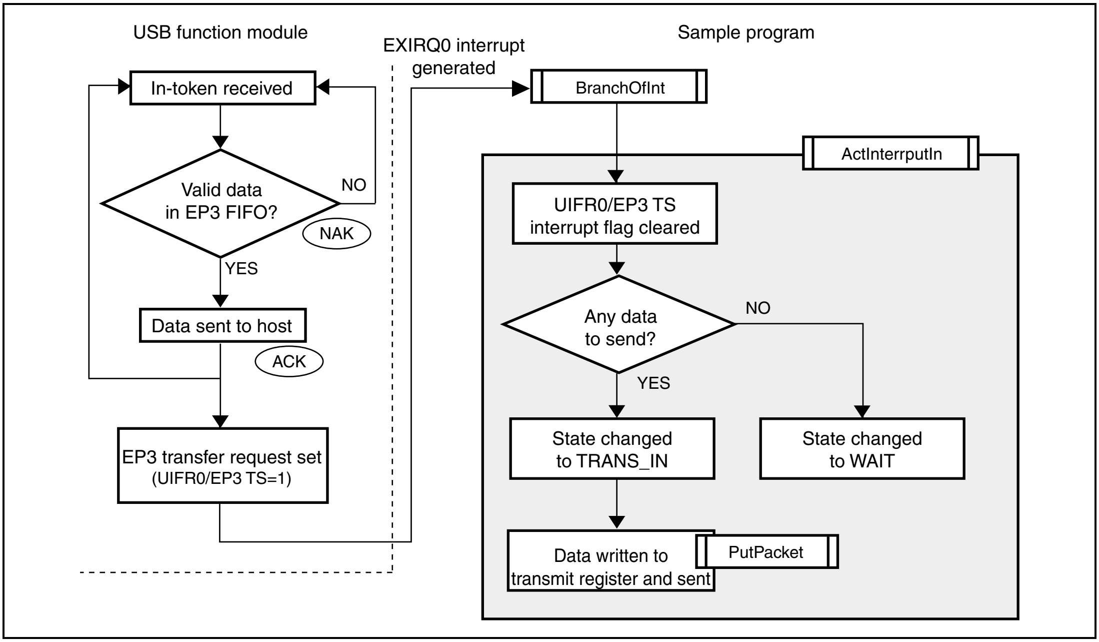


Figure 5.14 Interrupt-In Transfer

RENESAS

5.8 Mouse Data Generation

As no mouse can be connected to the MS2218CP, the sample program generates pseudo data (HID data) of the USB mouse by using SW5 to SW12 on the MS2218CP to emulate USB mouse operation.

To generate HID data, a 16-bit timer interrupt in the H8S/2218 is used to perform key scan for SW5 to SW12 on the MS2218CP.

- Pressing SW5 generates data for the left mouse button pressed
- Pressing SW6 generates data for the right mouse button pressed
- Pressing SW8 reads pointer movement data from the data table and demonstrates pointer movements (demonstration mode)
- Pressing SW9 generates data for pointer movement downward in the Y direction
- Pressing SW10 generates data for pointer movement upward in the Y direction
- Pressing SW11 generates data for pointer movement toward right in the X direction
- Pressing SW12 generates data for pointer movement toward left in the X direction

The data generated by these switches is passed to the ActMakeHidData function, and the HID data is sent to the host PC by using interrupt transfer. Figure 5.15 shows HID data generation of the sample program.

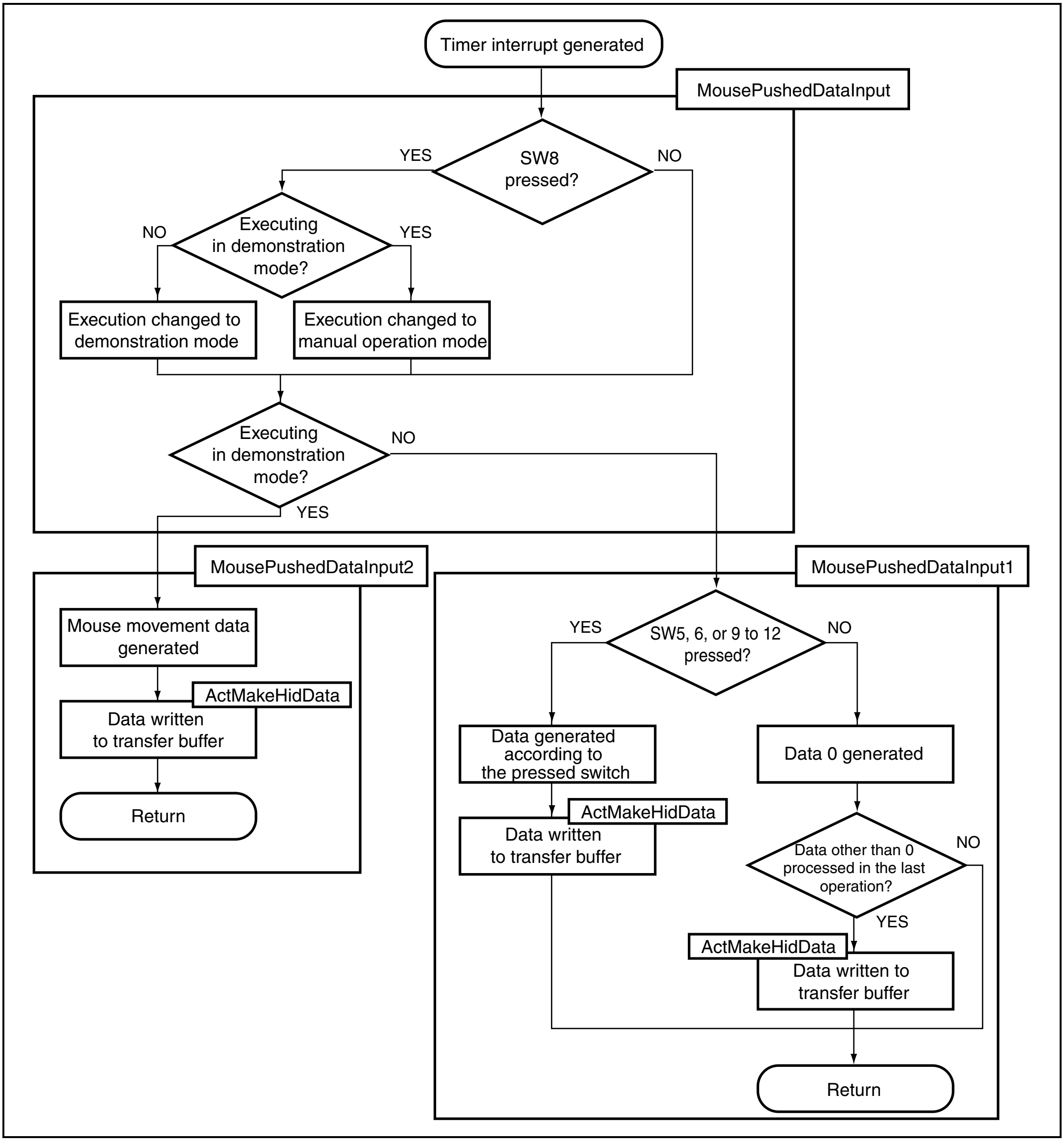


Figure 5.15 Mouse Data Generation

Section 6 Analyzer Data

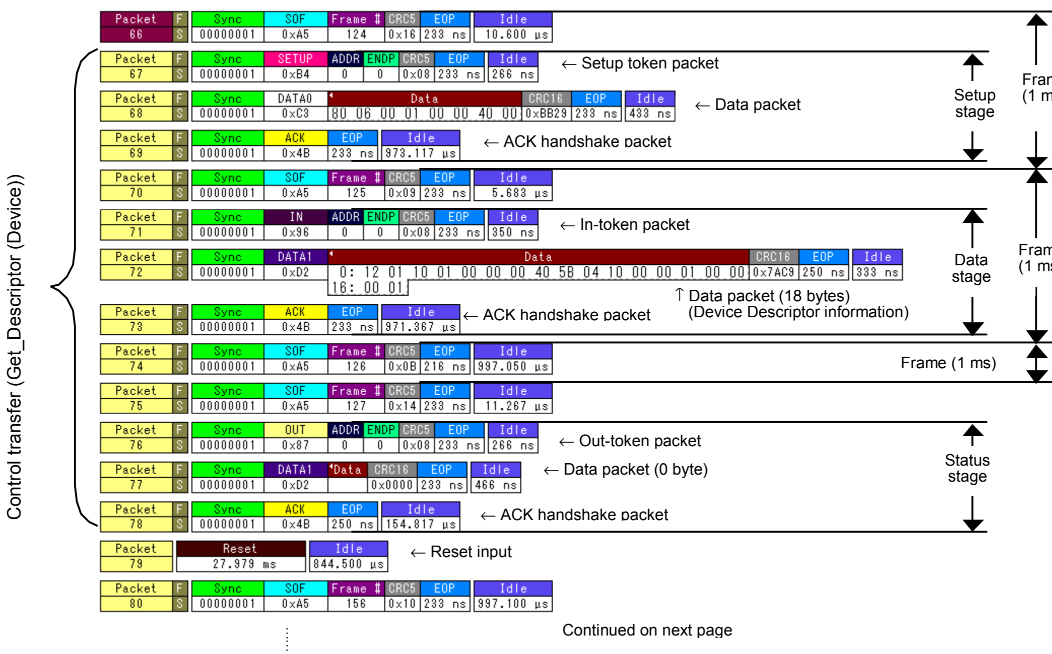
In this section, we look at how measurement is carried out with the USB Advisor, a USB protocol analyzer manufactured by CATC (http://www.catc.com), using the USB function module in the H8S/2218, and at what happens to the data as it actually flows along the bus. The following gives the description for control transfer when a device is connected and interrupt-in transfer of HID data as examples.

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

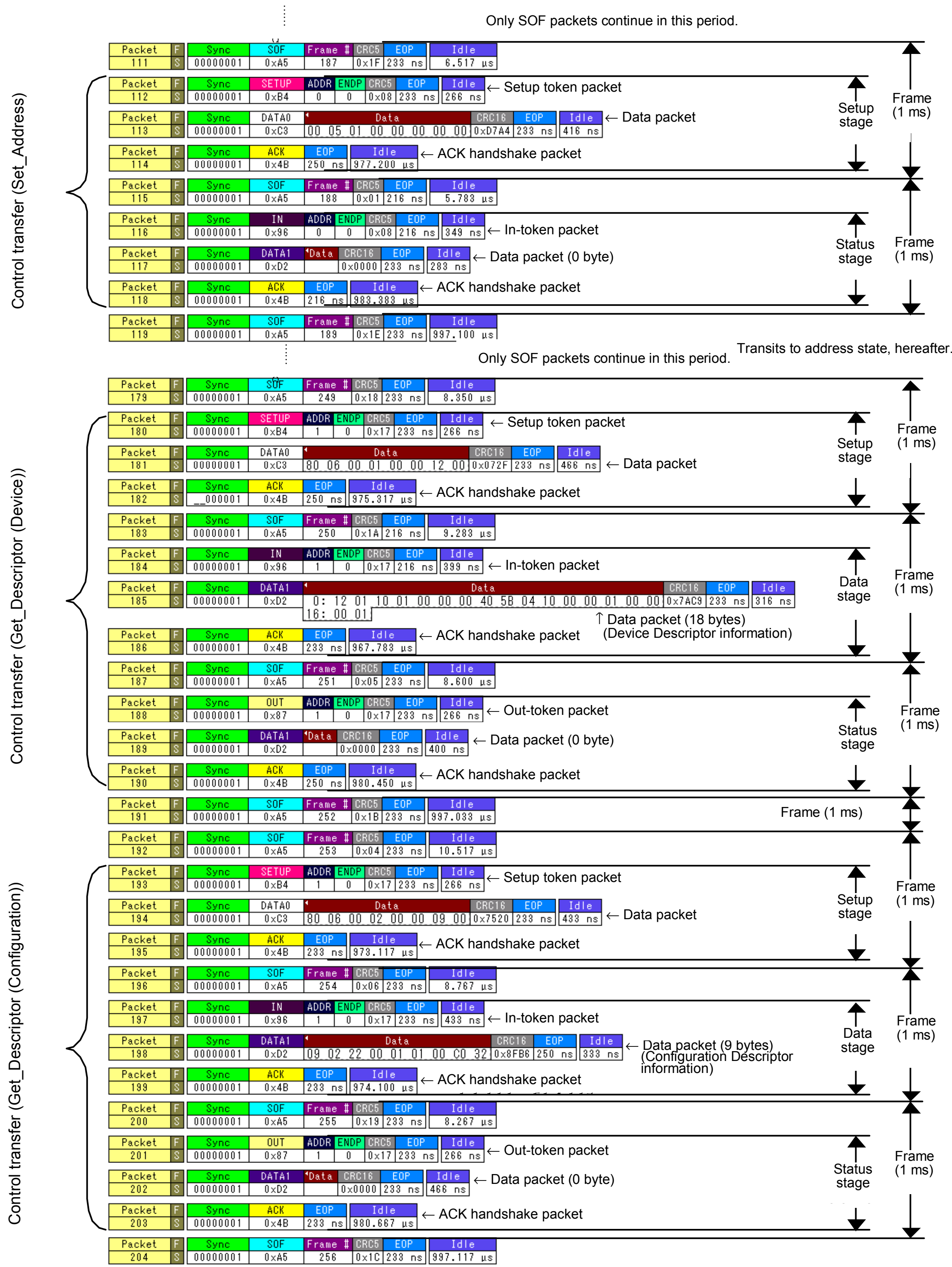
6.1 Control Transfer when Device is Connected

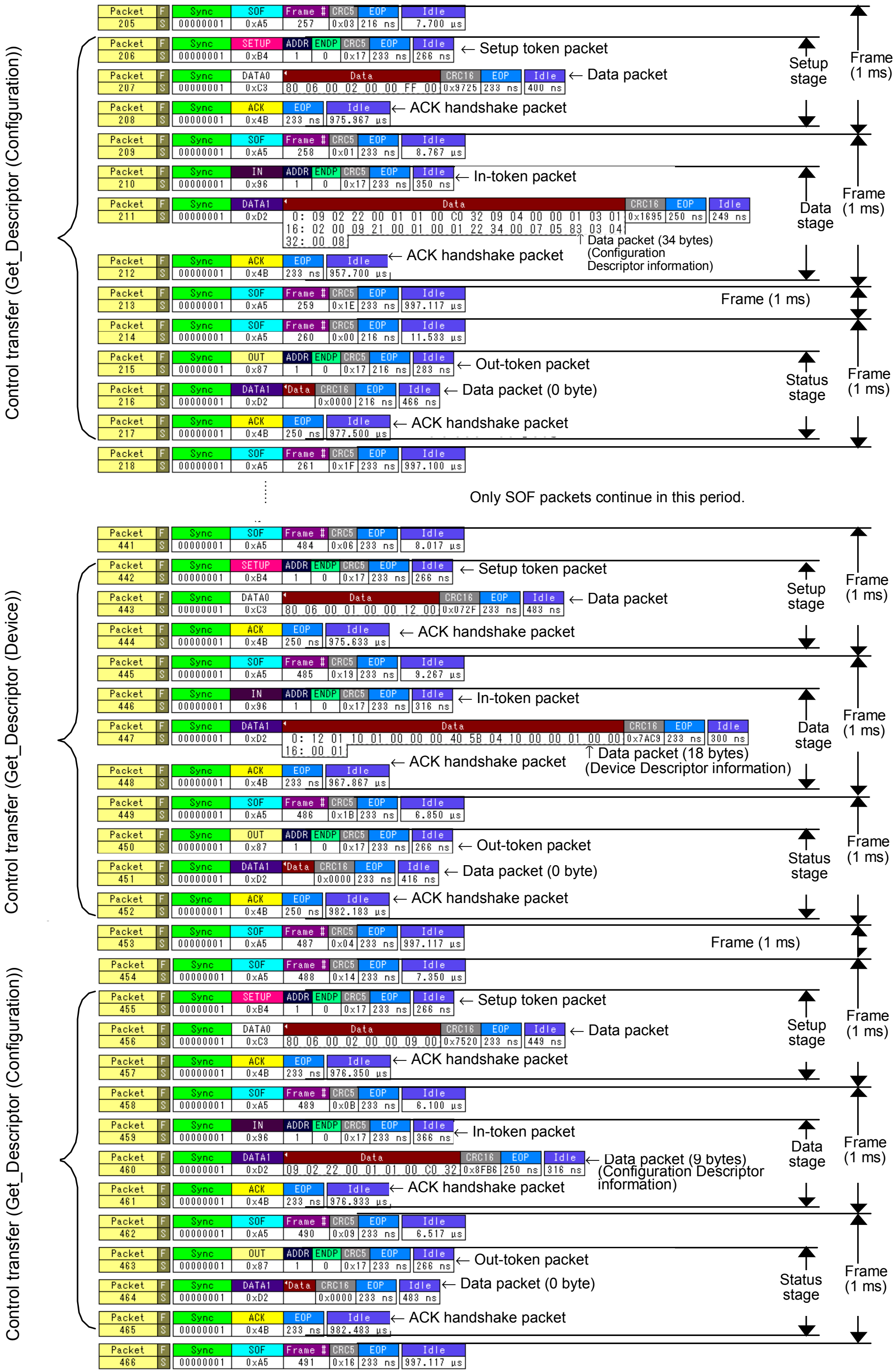
Figure 6.1 shows the measurement made, with a device connected to the host controller, while shifting from the power-on state (the power is supplied to Vbus) until the configuration state (device is ready for being used).

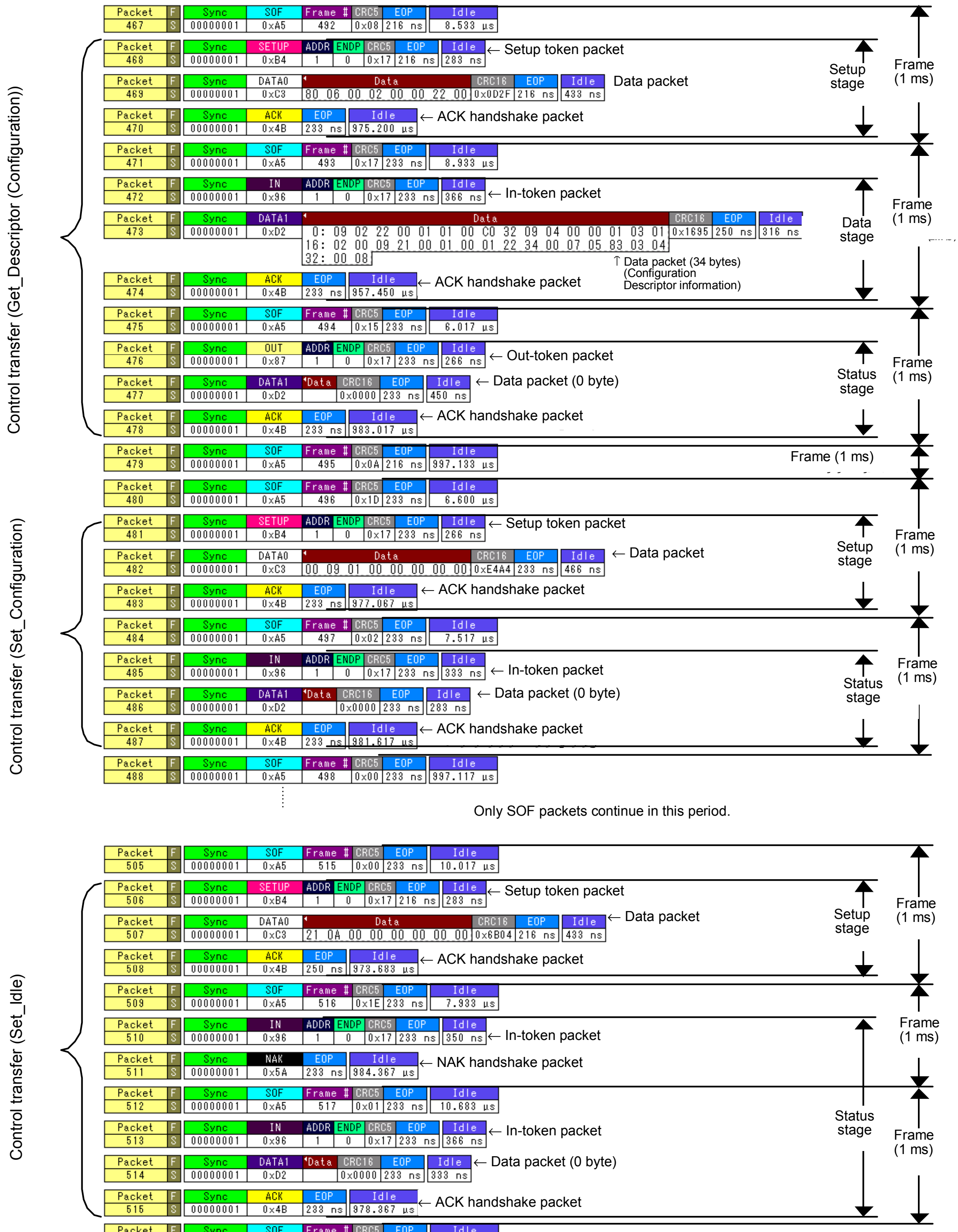
Though the packet scheduling may differ depending on the host controller, the command flow to the configuration state is always the same.



Continued on next page


Only SOF packets continue in this period.
Packet 111 | Sync 00000001 | SOF 0xA5 | Frame # 187 | CRC5 0x1F | EOP 233 ns | Idle 6.517 µs
Control transfer (Set_Address)
Packet 112 | Sync 00000001 | SETUP 0xB4 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 233 ns | Idle 266 ns ← Setup token packet
Packet 113 | Sync 00000001 | DATA0 0xC3 | Data 00 05 01 00 00 00 00 00 | CRC16 0xD7A4 | EOP 233 ns | Idle 416 ns ← Data packet
Packet 114 | Sync 00000001 | ACK 0x4B | EOP 250 ns | Idle 977.200 µs ← ACK handshake packet
Setup stage
Frame (1 ms)
Packet 115 | Sync 00000001 | SOF 0xA5 | Frame # 188 | CRC5 0x01 | EOP 216 ns | Idle 5.783 µs
Packet 116 | Sync 00000001 | IN 0x96 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 216 ns | Idle 349 ns ← In-token packet
Packet 117 | Sync 00000001 | DATA1 0xD2 | Data | CRC16 0x0000 | EOP 233 ns | Idle 283 ns ← Data packet (0 byte)
Packet 118 | Sync 00000001 | ACK 0x4B | EOP 216 ns | Idle 983.383 µs ← ACK handshake packet
Status stage
Frame (1 ms)
Packet 119 | Sync 00000001 | SOF 0xA5 | Frame # 189 | CRC5 0x1E | EOP 233 ns | Idle 997.100 µs
Transits to address state, hereafter.
Only SOF packets continue in this period.
Control transfer (Get_Descriptor (Device))
Packet 179 | Sync 00000001 | SOF 0xA5 | Frame # 249 | CRC5 0x18 | EOP 233 ns | Idle 8.350 µs
Packet 180 | Sync 00000001 | SETUP 0xB4 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 233 ns | Idle 266 ns ← Setup token packet
Packet 181 | Sync 00000001 | DATA0 0xC3 | Data 80 06 00 01 00 00 12 00 | CRC16 0x072F | EOP 233 ns | Idle 466 ns ← Data packet
Packet 182 | Sync __000001 | ACK 0x4B | EOP 250 ns | Idle 975.317 µs ← ACK handshake packet
Setup stage
Frame (1 ms)
Packet 183 | Sync 00000001 | SOF 0xA5 | Frame # 250 | CRC5 0x1A | EOP 216 ns | Idle 9.283 µs
Packet 184 | Sync 00000001 | IN 0x96 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 216 ns | Idle 399 ns ← In-token packet
Packet 185 | Sync 00000001 | DATA1 0xD2 | Data 0: 12 01 10 01 00 00 00 40 5B 04 10 00 00 01 00 00 16: 00 01 | CRC16 0x7AC9 | EOP 233 ns | Idle 316 ns
↑ Data packet (18 bytes)
(Device Descriptor information)
Packet 186 | Sync 00000001 | ACK 0x4B | EOP 233 ns | Idle 967.783 µs ← ACK handshake packet
Data stage
Frame (1 ms)
Packet 187 | Sync 00000001 | SOF 0xA5 | Frame # 251 | CRC5 0x05 | EOP 233 ns | Idle 8.600 µs
Packet 188 | Sync 00000001 | OUT 0x87 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 233 ns | Idle 266 ns ← Out-token packet
Packet 189 | Sync 00000001 | DATA1 0xD2 | Data | CRC16 0x0000 | EOP 233 ns | Idle 400 ns ← Data packet (0 byte)
Packet 190 | Sync 00000001 | ACK 0x4B | EOP 250 ns | Idle 980.450 µs ← ACK handshake packet
Status stage
Frame (1 ms)
Packet 191 | Sync 00000001 | SOF 0xA5 | Frame # 252 | CRC5 0x1B | EOP 233 ns | Idle 997.033 µs
Frame (1 ms)
Packet 192 | Sync 00000001 | SOF 0xA5 | Frame # 253 | CRC5 0x04 | EOP 233 ns | Idle 10.517 µs
Control transfer (Get_Descriptor (Configuration))
Packet 193 | Sync 00000001 | SETUP 0xB4 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 233 ns | Idle 266 ns ← Setup token packet
Packet 194 | Sync 00000001 | DATA0 0xC3 | Data 80 06 00 02 00 00 09 00 | CRC16 0x7520 | EOP 233 ns | Idle 433 ns ← Data packet
Packet 195 | Sync 00000001 | ACK 0x4B | EOP 233 ns | Idle 973.117 µs ← ACK handshake packet
Setup stage
Frame (1 ms)
Packet 196 | Sync 00000001 | SOF 0xA5 | Frame # 254 | CRC5 0x06 | EOP 233 ns | Idle 8.767 µs
Packet 197 | Sync 00000001 | IN 0x96 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 233 ns | Idle 433 ns ← In-token packet
Packet 198 | Sync 00000001 | DATA1 0xD2 | Data 09 02 22 00 01 01 00 C0 32 | CRC16 0x8FB6 | EOP 250 ns | Idle 333 ns ← Data packet (9 bytes) (Configuration Descriptor information)
Packet 199 | Sync 00000001 | ACK 0x4B | EOP 233 ns | Idle 974.100 µs ← ACK handshake packet
Data stage
Frame (1 ms)
Packet 200 | Sync 00000001 | SOF 0xA5 | Frame # 255 | CRC5 0x19 | EOP 233 ns | Idle 8.267 µs
Packet 201 | Sync 00000001 | OUT 0x87 | ADDR 1 | ENDP 0 | CRC5 0x17 | EOP 233 ns | Idle 266 ns ← Out-token packet
Packet 202 | Sync 00000001 | DATA1 0xD2 | Data | CRC16 0x0000 | EOP 233 ns | Idle 466 ns ← Data packet (0 byte)
Packet 203 | Sync 00000001 | ACK 0x4B | EOP 233 ns | Idle 980.667 µs ← ACK handshake packet
Status stage
Frame (1 ms)
Packet 204 | Sync 00000001 | SOF 0xA5 | Frame # 256 | CRC5 0x1C | EOP 233 ns | Idle 997.117 µs


Control transfer (Get_Descriptor (Configuration))
← Setup token packet
← Data packet
← ACK handshake packet
← In-token packet
Data packet (34 bytes)
(Configuration Descriptor information)
← Out-token packet
← Data packet (0 byte)
Setup stage
Data stage
Status stage
Frame (1 ms)
Only SOF packets continue in this period.
Control transfer (Get_Descriptor (Device))
Data packet (18 bytes)
(Device Descriptor information)
Control transfer (Get_Descriptor (Configuration))
Data packet (9 bytes)
(Configuration Descriptor information)



Continued on next page

RENESAS

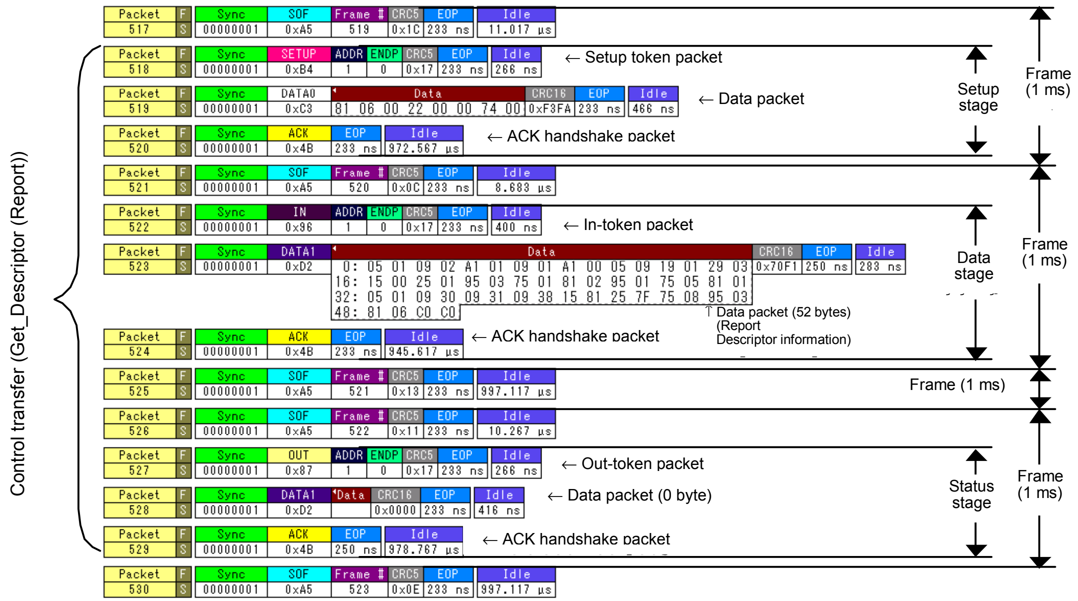


Figure 6.1 Control Transfer when Device is Connected

6.2 Interrupt-In Transfer of HID Data

Figure 6.2 shows the measurement results when HID data is sent from the device to the USB host controller through interrupt-in transfer. In response to the interrupt-in transfer from the USB host controller, the device returns a NAK if no data can be sent. If there is data to be sent, the device sends 4-byte HID data. On receiving HID data, the USB host controller issues an ACK.

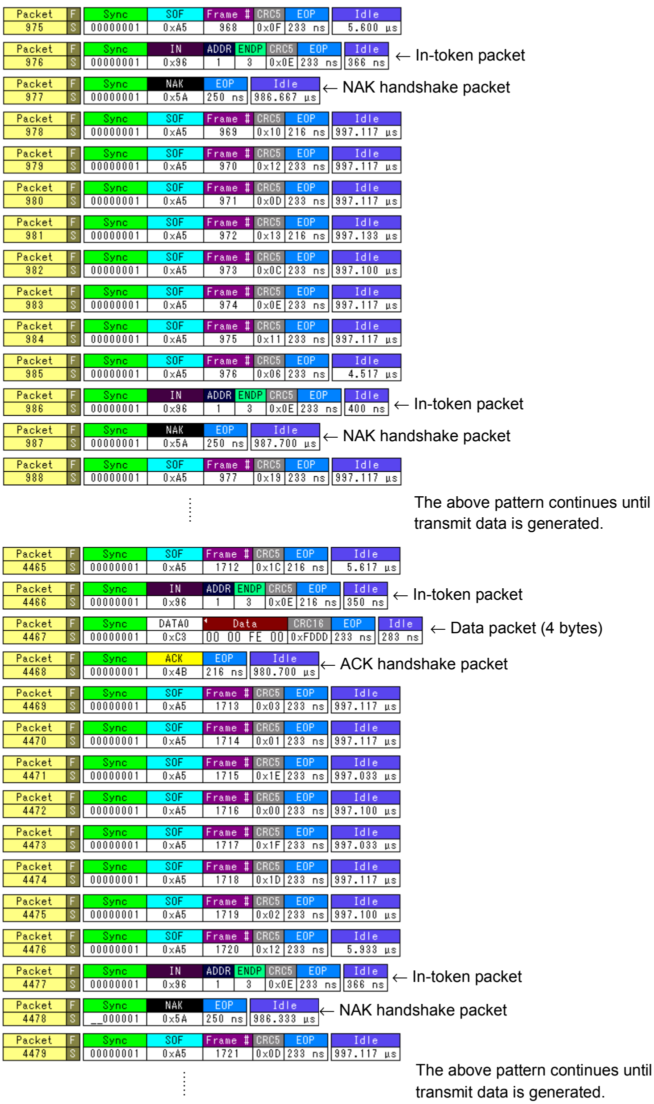


Figure 6.2 Interrupt-In Transfer of HID Data

H8S/2218 USB Function Module
Human Interface Devices (HID) Class Application Note

Publication Date: Rev.1.00, October 20, 2003
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Technical Documentation & Information Department
Renesas Kodaira Semiconductor Co., Ltd.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS SALES OFFICES

http://www.renesas.com

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500 Fax: <1> (408) 382-7501

Renesas Technology Europe Limited.
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, United Kingdom
Tel: <44> (1628) 585 100, Fax: <44> (1628) 585 900

Renesas Technology Europe GmbH
Dornacher Str. 3, D-85622 Feldkirchen, Germany
Tel: <49> (89) 380 70 0, Fax: <49> (89) 929 30 11

Renesas Technology Hong Kong Ltd.
7/F., North Tower, World Finance Centre, Harbour City, Canton Road, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2375-6836

Renesas Technology Taiwan Co., Ltd.
FL 10, #99, Fu-Hsing N. Rd., Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology (Shanghai) Co., Ltd.
26/F., Ruijin Building, No.205 Maoming Road (S), Shanghai 200020, China
Tel: <86> (21) 6472-1001, Fax: <86> (21) 6415-2952

Renesas Technology Singapore Pte. Ltd.
1, Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

H8S/2218 USB Function Module Human Interface Devices (HID) Class

Application Note

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ06B0212-0100Z