| CUSTOMER NOTIFICATION | SUD-DT-03-0137-E |
|---|---|
| | April 14, 2003 |
| VR4181A Application Note:<br>Usage of VR4181A On-Chip USBF Control Unit | Toru Henmi, Senior System Integrator<br>Microcomputer Group<br>2nd Solutions Division<br>Solutions Operations Unit<br>NEC Electronics Corporation |

CP(K), O

Thank you for your continued support of NEC Electronics products.
This is to inform you of the following items related to the VR4181A on-chip USBF control unit.

♦ Details of report

This is the Application Note for the on-chip USB function control unit provided in the VR4181A ($\mu$PD30181A and 30181AY).

This document should be used as a reference when creating the VR4181A USBFU device driver, and contains the following information.

- Outline of VR4181A USBFU
- VR4181A USBFU sample program

  Mouse device driver software for HID class devices

♦ Target products

- All ranks of $\mu$PD30181AF1-131-GA3 (all versions)
- All ranks of $\mu$PD30181AYF1-131-GA3 (all versions)

Exporting this product or equipment that includes this product may require a governmental license from the U.S.A. for some countries because this product utilizes technologies limited by the export control regulations of the U.S.A.

- **The information in this document is current as of March, 2003. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "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.
  "Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

# CONTENTS

# 1 INTRODUCTION

This document describes the usage of the HID (human interface device) class mouse device which uses the VR4181A on-chip USB function control unit (hereafter referred to as USBFU). Please reference this document when creating the VR4181A USBFU device driver.

This document describes the configuration of the VR4181A on-chip USBFU taking HID class mouse device communication as an example. The contents of this document and software are simply for showing examples of USBFU usage, and are not guaranteed in actual operation. Modification and addition are required when using the device driver for general purposes.

Also refer to the following documents for development.

[Related Documents]

- Universal Serial Bus Specification Revision 1.1
- Universal Serial Bus Device Class Definition for HID Devices
- VR4181A Hardware User's Manual (U16049EJ1)
- Ostrich Hardware specification (VR4181A solution board specification)

# 2 GENERAL

This document describes the usage of the VR4181A USBFU and examples for creating the device driver.

The features of the VR4181A on-chip function control unit are shown below.

- Includes USBFU conforming to USB1.1
- Full-speed transfer (12 Mbps) supported
- Provided with bus master function, which enables DMA transfer with the local unit
- Includes endpoints for transfer
  - Endpoint for control transfer (for transmission): 64-byte FIFO
  - Endpoint for interrupt transfer (for transmission): 64-byte FIFO
  - Endpoint for bulk transfer (for transmission): 64 x 2-byte FIFO
  - Endpoint for control/interrupt/bulk/isochronous transfer (for reception): 12-byte FIFO (common)
- Conforming to the USB communication device class

Figure 1 shows the system configuration.

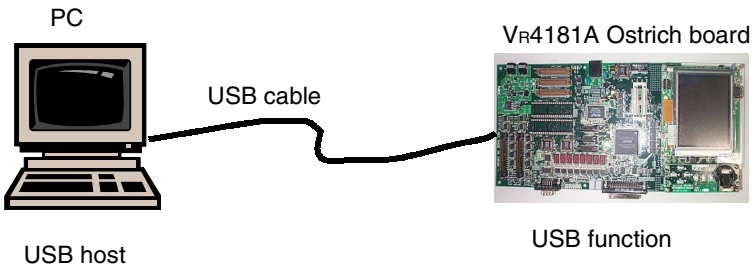



**Figure 1. System Configuration**

This system is configured by the NEC Electronics VR4181A solution board (Ostrich board) with the VR4181A on board and a personal computer with Windows 2000 as the OS.

In this system, USB mouse device position data is transmitted from the VR4181A USBFU to the host PC. The USB-HID class device driver that is supplied with Windows 2000 as standard and USB mouse device driver are used in the host PC, which enables recognition of mouse pointer movement on the host PC display.

# 3 OUTLINE OF USB

## 3.1 USB Device Framework

To implement Plug &Play, the procedure from connecting the USB cable to the end of configuration is defined in detail in the USB specifications. This section explains the procedure.

### 3.1.1 State of device

Figure 2 shows the states of the USB device. Users can use the device only when the device is in the configuration state.

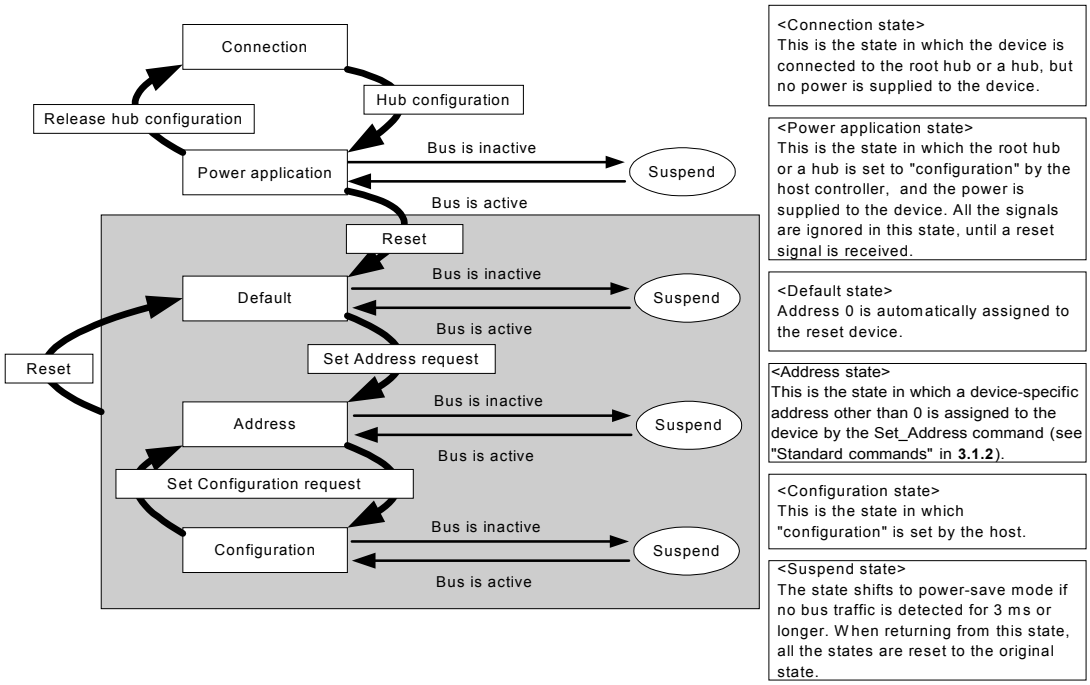



**Figure 2. USB Device States**

### 3.1.2 Device requests

To shift to the configuration state, the device must respond to a command issued by the host controller. The command issued by the host controller is called a device request, and its format is defined in the USB specifications. The host controller issues a device request in the setup stage of control transfer.

The device requests are classified into the following three types as shown below.

- Standard commands

These are the commands defined in the USB Specification, and must in principle be supported by all the devices. Table 1 shows a list of the standard commands.

See the USB Specification for details of the standard commands.

**Table 1. List of Standard Commands**

| Command Name | Function | Data Stage | Data Stage Direction |
|---|---|---|---|
| Clear_Feature (Endpoint_Halt) | Releases endpoint stall state | Not provided | – |
| Clear_Feature (Device_Remote_Wakeup) | Releases remote wakeup of device | Not provided | – |
| Get_Configuration | Acquires configuration information | Provided | In |
| Get_Descriptor (Device) | Acquires device descriptor information | Provided | In |
| Get_Descriptor (Config) | Acquires configuration descriptor information | Provided | In |
| Get_Descriptor (String) | Acquires string descriptor information | Provided | In |
| Get_Interface | Acquires interface information | Provided | In |
| Get_Status (Device) | Acquires device status information | Provided | In |
| Get_Status (Interface) | Acquires interface status information | Provided | In |
| Get_Status (EndPoint) | Acquires endpoint status information | Provided | In |
| Set_Address | Sets device address | Not provided | – |
| Set_Descriptor (Device) | Sets device descriptor | Provided | Out |
| Set_Descriptor (Config) | Sets configuration descriptor | Provided | Out |
| Set_Descriptor (String) | Sets string descriptor | Provided | Out |
| Set_Configuration | Sets configuration | Not provided | – |
| Set_Feature (EndPoint_Halt) | Sets the endpoint to stall state | Not provided | – |
| Set_Feature (Device_Remote_Wakeup) | Sets the device to remote wakeup state | Not provided | – |
| Set_Interface | Sets interface | Not provided | – |
| Synch_Frame | Reports a specific frame number to the endpoint when performing isochronous transfer (only when a specific number is required) | Provided | Out |

- Class commands

  The class commands other than the hub are determined by groups such as corporations, and authorized by the USB-IF (USB Implementers Forum). The classes include the audio class, common class, HID (Human Interface Device) class, and printer class.

- Vendor commands

  The vendor commands can be defined freely by the device developer. However, their format must be consistent with other commands.

## 3.2 Descriptors

The USB device has “descriptor” information which indicates the type, features, and attribute of the device. The host controller recognizes which device is connected to the bus based on the acquired descriptor information.

Four descriptors: device, configuration, interface, and endpoint are provided in a standard USB device.

Tables 2 to 5 show the descriptors.

**Table 2. Device Descriptor**

| Field | Size (Bytes) | Description |
|---|---|---|
| bLength | 1 | Descriptor size (fixed to 0x12 ) |
| bDescriptorType | 1 | Descriptor type (fixed to 0x01) |
| bcdUSB | 2 | USB version indicated by BCD |
| bDeviceClass | 1 | Class code<br>0: No class<br>0xFF: Vendor class<br>1 to 0xFE: Specific class |
| bDeviceSubClass | 1 | Sub-class code |
| bDeviceProtocol | 1 | Protocol code<br>0: Specific protocol not used<br>0xFF: Vendor-specific protocol |
| bMaxPacketSize0 | 1 | Maximum packet size of endpoint 0 |
| idVendor | 2 | Vendor ID (assigned to manufacturer by USB-IF[Note]) |
| idProduct | 2 | Product ID (assigned to each device by manufacturer) |
| bcdDevice | 2 | Device version indicated by BCD |
| iManufacturer | 1 | Index to string descriptor indicating manufacturer |
| iProduct | 1 | Index to string descriptor indicating device name |
| iSerialNumber | 1 | Index to string descriptor indicating serial number of the device |
| bNumConfigurations | 1 | Configurable number |

**Note** USB Implementers Forum

**Table 3. Configuration Descriptor**

| Field | Size (Bytes) | Description |
|---|---|---|
| bLength | 1 | Descriptor size (fixed to 0x09) |
| bDescriptorType | 1 | Descriptor type (fixed to 0x02) |
| wTotalLength | 2 | Total length of descriptors |
| bNumInterface | 1 | Number of interfaces in this descriptor |
| bConfigurationValue | 1 | An argument value (1 or larger) for selecting the configuration descriptor using Set_Configuration |
| iConfiguration | 1 | Index to string descriptor |
| bmAttributes | 1 | Power supply of device<br>Bit 7: Reserved (1)<br>Bit 6: On-chip power supply<br>Bit 5: Remote wakeup<br>Bit 4 to 0: Reserved (0) |
| MaxPower | 1 | Specifies the max. bus current consumption in 2 mA units |

**Table 4. Interface Descriptor**

| Field | Size (Bytes) | Description |
|---|---|---|
| bLength | 1 | Descriptor size (fixed to 0x09) |
| bDescriptorType | 1 | Descriptor type (fixed to 0x04) |
| bInterfaceNumber | 1 | 0–base index number for representing this interface in the configuration |
| bAlternateSetting | 1 | An argument value for selecting an alternate setting using Set_Interface |
| bNumEndpoints | 1 | The number of endpoints in a device (except for endpoint 0 ) |
| bInterfaceClass | 1 | Class code<br>0: No class<br>0xFF: Vendor class<br>1 to 0xFE: Specific class |
| bInterfaceSubClass | 1 | Sub-class code |
| bInterfaceProtocol | 1 | Protocol code<br>0: Specific protocol not used<br>0xFF: Vendor-specific protocol |
| iInterface | 1 | Index to string descriptor for representing this interface |

**Table 5. Endpoint Descriptor**

| Field | Size (Bytes) | Description |
|---|---|---|
| bLength | 1 | Descriptor size (fixed to 0x07) |
| bDescriptorType | 1 | Descriptor type (fixed to 0x05) |
| bEndpointAddress | 1 | Endpoint address<br>Bit 7: Direction (0: OUT, 1: IN)<br>Bit 6 to 4: Reserved (0)<br>Bits 3 to 0: Endpoint number |
| bmAttributes | 1 | Transfer mode of endpoint<br>Bits 7 to 2: Reserved (0)<br>Bits 1 and 0: Transfer mode (0: Control, 1: Isochronous, 2: Bulk, 3: Interrupt) |
| wMaxPacketSize | 2 | Maximum packet size |
| bInterval | 1 | Polling interval can be specified in 1 ms units.<br>Specify 1 for isochronous transfer<br>This bit is ignored when bulk or control transfer is performed |

# 4 OUTLINE OF VR4181A ON-CHIP USBFU

## 4.1 VR4181A USBFU Operation

This section explains the operation of the VR4181A on-chip USBFU. Figure 3 shows the block configuration elements of the USBFU and its connection with peripheral circuits.

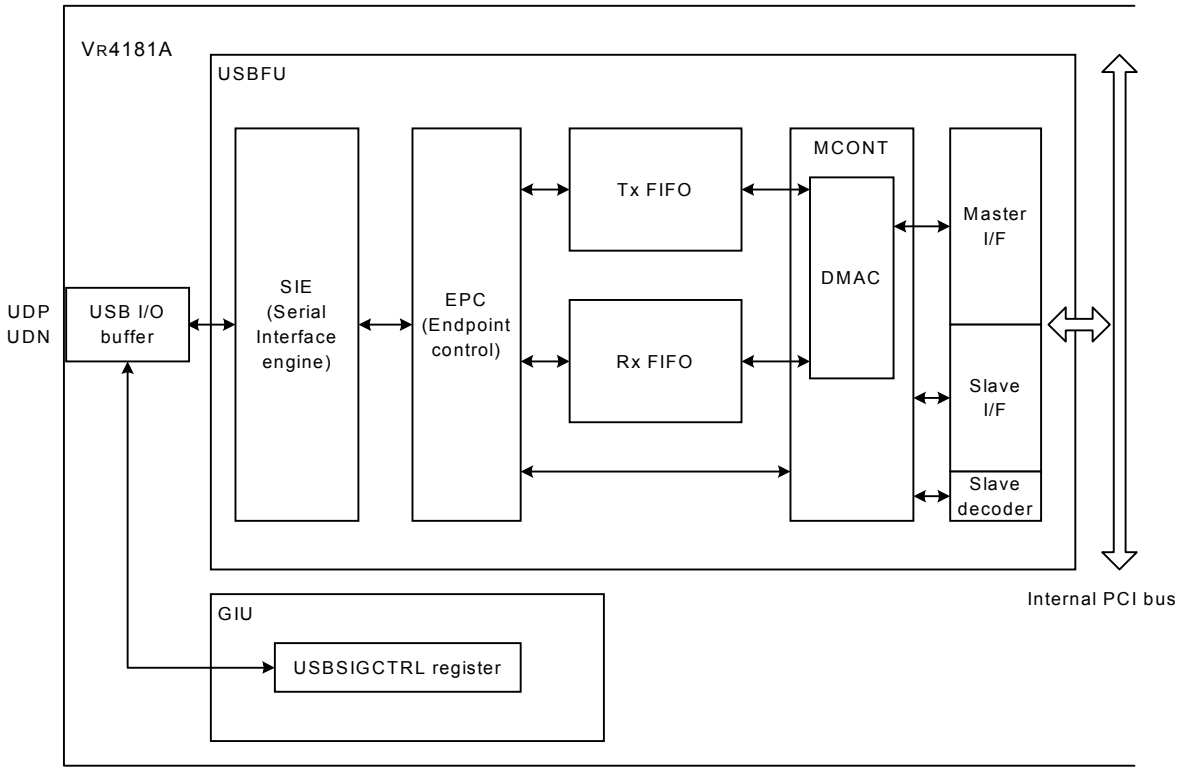



**Figure 3. Block Diagram of VR4181A USBFU**

The VR4181A USBFU transmits to or receives data from the USB host using the following functions.

- Transmit/receive mailbox
- Transmit packet descriptor and transmit buffer descriptor
- Receive packet descriptor and receive buffer descriptor

(1) Transmit/receive mailbox

The transmit/receive mailbox is secured in the memory. The VR4181A USBFU updates the transmit/receive status each time a data segment is transmitted or received.

See **13.2.2 Mailbox** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(2) Transmit packet descriptor and transmit buffer descriptor

The transmit descriptor includes two types: a transmit packet descriptor and transmit buffer descriptor.

The transmit packet descriptor specifies the number of bytes of transmit data and its start address in the relevant transmit buffer directory using the USB Command register (offset: 0x40) and USB Command Extension register (offset: 0x44).

The transmit buffer directories are secured in the memory, and include transmit buffer descriptors. Transmit buffer directories can be linked to each other using the link pointer.

Transmit buffer descriptors re secured in the memory and specify the start address and size of the transmit data buffer. The last buffer descriptor is specified by setting the L (Last) bit to 1.

The software divides the data segment to be transmitted into USB packets and stores them in the memory using the transmit data buffer.

Figure 4 shows the configuration of the transmit descriptor and transmit data buffer.

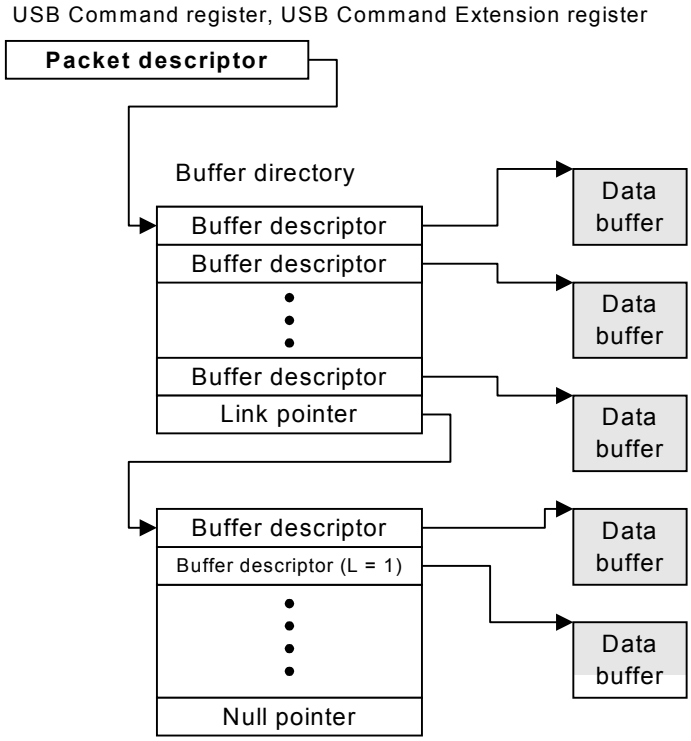



**Figure 4. Specification of Transmit Data Buffer Using Transmit Descriptors**

See **13.2.3 Structure of transmit buffer and descriptor** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(3) Receive packet descriptor and receive buffer descriptor

The receive descriptor includes two types: a receive packet descriptor (Pooln descriptor (n = 0 to 2)) and receive buffer descriptor.

The receive packet descriptor (Pooln descriptor (n = 0 to 2)) specifies the number of receive buffer directories added to the receive pool and their start addresses using the USB Command register (offset: 0x40) and USB Command Extension register (offset: 0x44).

The receive buffer directories are secured in the memory, and include receive buffer descriptors. Receive buffer directories can be linked to each other using the link pointer.

Receive buffer descriptors are secured in the memory and specify the start address and size of the receive data buffer. The last buffer descriptor is specified by setting the L (Last) bit to 1.

The USBFU of the VR4181A divides the receive data segment into USB packets and stores them in the memory using the receive data buffer.

Figure 5 shows the configuration of the receive descriptor and receive data buffer.

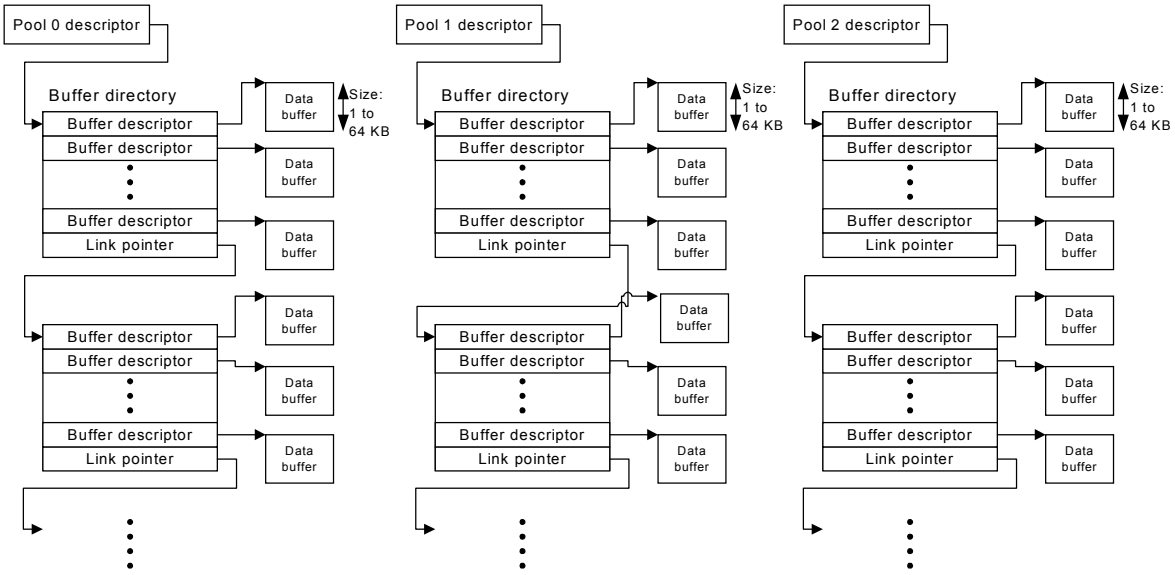



**Figure 5. Specification of Receive Data Buffer Using Receive Descriptors**

See **13.2.4 Structure of receive buffer and descriptor** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

## 4.2 Endpoint Configuration

The VR4181A on-chip USBFU is provided with six endpoints. Table 6 shows the configuration of the endpoints in the USBFU.

**Table 6. Endpoint Configuration**

| Endpoint Name | FIFO Size (Bytes) | Transfer Type | Transfer Direction |
|---|---|---|---|
| Endpoint 0 | 64 (transmission),<br>64 x 2 (reception) Note | Control transfer | Transmission/reception (IN/OUT transaction) |
| Endpoint 1 | 64 x 2 | Isochronous transfer | Transmission (IN transaction) |
| Endpoint 2 | 64 x 2Note | Isochronous transfer | Reception (OUT transaction) |
| Endpoint 3 | 64 x 2 | Bulk transfer | Transmission (IN transaction) |
| Endpoint 4 | 64 x 2Note | Bulk transfer | Reception (OUT transaction) |
| Endpoint 5 | 64 | Interrupt transfer | Transmission (IN transaction) |
| Endpoint 6 | 64 x 2Note | Interrupt transfer | Reception (OUT transaction) |

**Note** The 128-byte FIFO is used commonly by the receive endpoints.

# 5 OUTLINE OF SAMPLE PROGRAM

This chapter explains the features of the sample program and its configuration.

This sample program operates by polling the status written in the transmit/receive mailbox.

**Note** Interrupt servicing can be performed based on the interrupt status written in USB General Status register 1 and USB General Status register 2 of the USBFU. For example, the interrupt status of endpoint 0 transfer end is indicated by the receive end interrupt status EP0RF and transmit end interrupt status EP0TF.

The features of this sample program are as follows.

- Control transfer can be performed.
- Data can be transmitted to the host controller by interrupt transfer.

## 5.1 Development Environment

This section explains the development environment used for developing this system. The following tools were used to develop this system.

| | |
|---|---|
| Target hardware: VR4181A Ostrich solution board | NEC Electronics |
| VR4181A-supporting N-Wire ICE: PARTNER-J | Kyoto Microcomputer |
| Compiler: GHS compiler 1.8.9 | Green Hills |

PC for USB host (Windows 2000)

USB cable (with Standard series-A connector and Mini-B connector)

Figure 6 shows the configuration of the development environment.

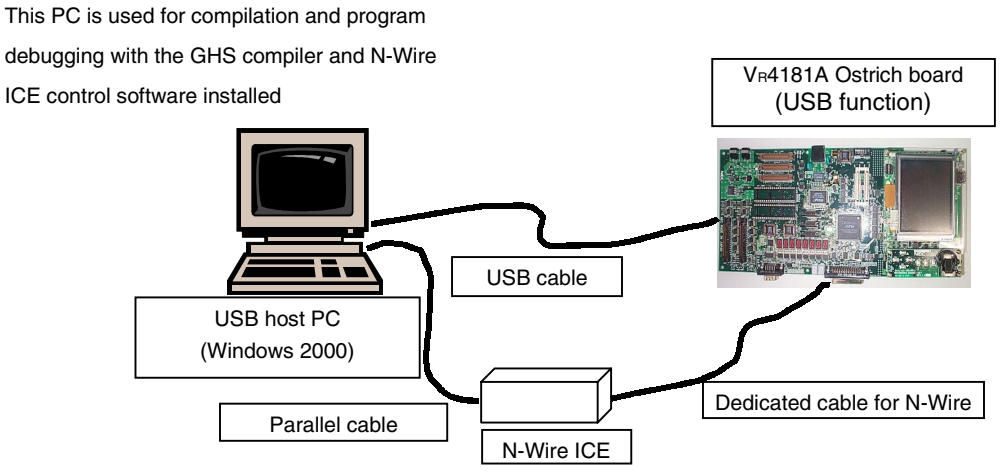



**Figure 6. Development Environment**

(1) VR4181A Ostrich board

Since the VR4181A Ostrich board is used in combination with the N-Wire ICE, the DIP switch setting of the VR4181A Ostrich board at shipment must be changed. Set SW1-1 on the CPU main board to ON before power application. It is not required to change other DIP switch settings.

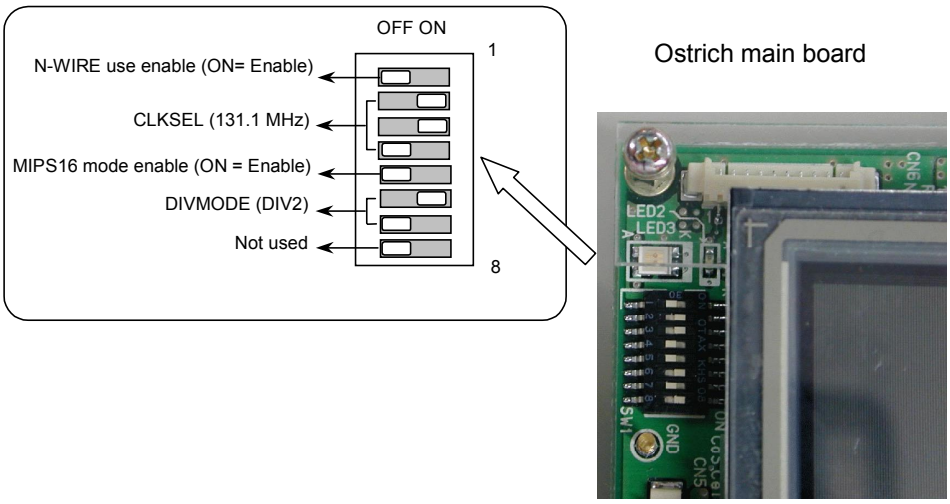



**Figure 7. CPU Main Board DIP Switch**

(2) USB host PC

A personal computer with a USB port and in which Windows 2000 is installed is used as the USB host. This system uses the USB-HID class device driver and USB mouse device driver that are included as standard in Windows 2000, so users are not required to install drivers.

(3) N-Wire ICE (Kyoto Microcomputer Partner-J)

Connect the system with the host PC using the parallel cable, and VR4181A Ostrich board using the dedicated cable for N-Wire. Control software must be installed in the host PC.

## 5.2 File Configuration

This sample program includes three source and four header files. Table 7 shows the file configuration.

**Table 7. File Configuration**

| File Name | Major Function |
|---|---|
| Iopciu.c | Initial setting of VR4181A USBFU |
| Iopciu_init.mip | Initial setting of VR4181A USBFU |
| Usbfunc.c | Initial setting of transmit/receive mailbox, descriptors, and buffers<br>Initial setting of USB device descriptor<br>Packet transmission/reception |
| Usbfunc.h | VR4181A USBFU register definition |
| Dsc_input.h | USB device descriptor definition used in USBF (example) |
| Dsc_table.h | USB device descriptor set value used in USBF (example) |
| Hid_class.h | HID descriptor set value used in USBF (example) |

**Note** The files Dsc_input.h, Dsc_table.h, and Hid_class.h simply show definition and setting value examples of the USB device descriptor; they are not used in this sample program.

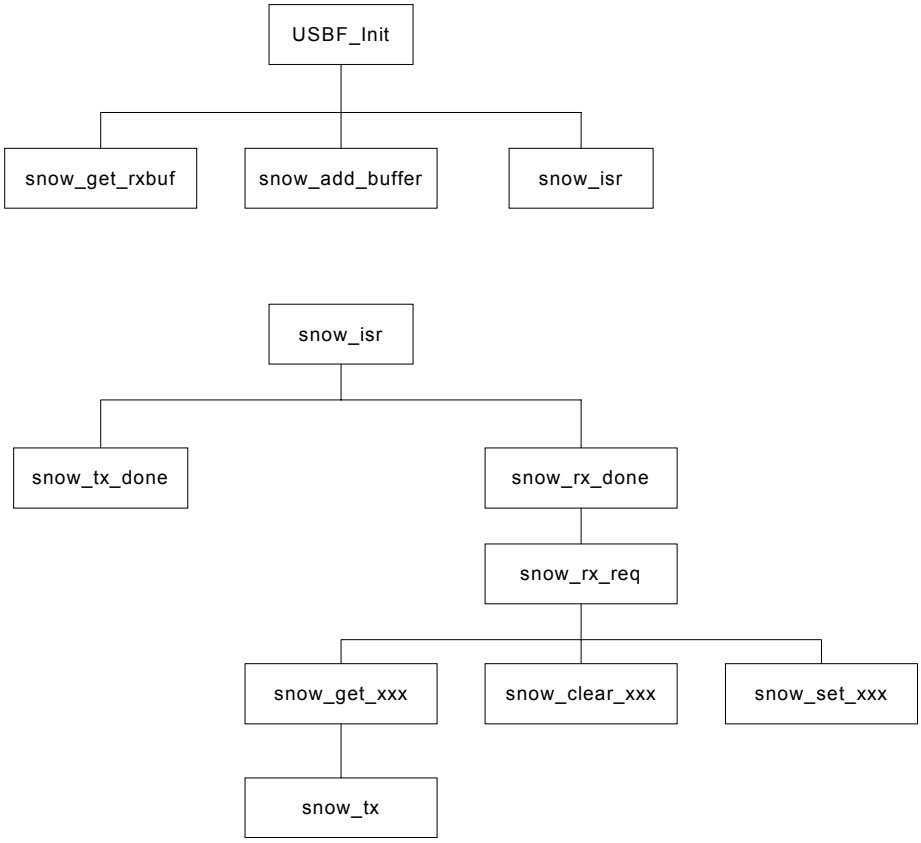
## 5.3 Function Features

Table 8 shows the functions included in the Usbfunc.c file and their features.

**Table 8. Usbfunc.c**

| Location File | Function Name | Features |
|---|---|---|
| Usbfunc.c | USBF_Init() | Initial setting of transmit/receive mailbox, descriptors, and buffers<br>Initial setting of USB device descriptor |
| | snow_get_rxbuf() | Securement of receive data buffer |
| | snow_add_buffer() | Addition of receive data buffer |
| | snow_isr() | Judgment of transmit/receive status<br>**Note** This sample program executes this function by polling. |
| | snow_rxreq() | Processing of device request from the USB host (control transfer, endpoint 0) |
| | snow_get_status() | Device request (Get_Status) processing |
| | snow_clear_feature() | Device request (Clear_Feature) processing |
| | snow_set_feature() | Device request (Set_Feature) processing |
| | snow_get_descriptor() | Device request (Get_Descriptor) processing |
| | snow_set_descriptor() | Device request (Set_Descriptor) processing |
| | snow_get_configuration() | Device request (Get_Configuration) processing |
| | snow_set_configuration() | Device request (Set_Configuration) processing |
| | snow_set_interface() | Device request (Set_Interface) processing |
| | snow_sync_frame() | Device request (Synch_Frame) processing |
| | snow_rxdone() | Receive end processing |
| | snow_tx() | Transmit processing |
| | snow_txdone() | Transmit end processing |

Figure 8 shows the relationship between the functions described in Table 8. The upper-side functions call the lower-side functions.




snow_get_xxx:

snow_get_status, snow_get_descriptor, snow_get_configuration, snow_get_interface

snow_clear_xxx:

snow_clear_feature, snow_set_feature, snow_set_configuration, snow_set_interface

snow_set_xxx:

snow_set_descriptor, snow_set_configuration

**Figure 8. Relationship Between Functions**

# 6 OPERATION OF SAMPLE PROGRAM

## 6.1 USBF Driver Operation Procedure

### 6.1.1 Initial settings on VR4181A USBFU

The following initial settings are required to access the operational registers in the VR4181A USBFU.

(1) Assignment of address window to SDRAM space and internal PCI bus space

The base address used for accessing the SDRAM space from the CPU and the area size can be set using the SDRAM register (offset: 0x00 0000).

The base address used for accessing the internal PCI bus space from the CPU and the area size can be set using the PCIW0 register (offset: 0x00 0060, for window 0) and PCIW1 register (0x00 0068, for window 1).

The base address used for accessing the internal system bus from the master device (USBFU in this sample program) of the internal PCI bus can be set using the BAR_INTCS register (offset: 0x00 0210), BAR_SDRAM register (offset: 0x00 0218), etc. See the description of the iopciu_init.mip file for examples of the BAR_xxx registers.

See **6.3 Assignment of Physical Addresses** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(2) Supplying clock to USBFU and internal PCI bridge (IOPCIU)

<1> Set the CMODE (1:0) bit of the PINMODED2 register in the GIU to 01 to enable the use of the CLK48 pin (line 48 of iopciu.c).

<2> Set the MSKUSBF48M bit of the CMUCLKMSK1 register in the CCU to 1 to supply CLK48 to the USBFU (line 53 of iopciu.c).

<3> Set the MSKUSBFPCI and MSKIOPCICLK bits of the CMUCLKMSK3 register in the CCU to 1 to supply PCIClock to the internal PCI bridge (IOPCIU) and USBF controller (line 54 of iopciu.c).

See **13.4.1 Initial settings (USBFU)** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(3) Resetting internal PCI bus

<1> Set the PCIWRST and PCICRST bits of the PCICTRL-H register in the internal PCI bridge (IOPCIU) to 1 while the clock is being supplied to the IOPCIU and USBFU to reset the units connected to the internal PCI bus (the PCICRST bit is automatically cleared to 0).

<2> Clear the PCIWRST bit of the PCICTRL-H register to 0 to release the warm reset of the internal PCI bus.

See the description of the iopciu_init.mip file for details of resetting the internal PCI bus.

See **13.4.1 Initial settings (USBFU)** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(4) USBFU pin setting

<1> Set the IENF and PSF bits of the USBSIGCTRL register in the GIU to 1. This activates the differential circuit of the UDN and UDP pins (line 50 of iopciu.c).

See **13.4.1 Initial settings (USBFU)** in **VR4181A Hardware User's Manual (U16049EJ1)** for details.

(5) Initialization of configuration space of the internal PCI bus

The VR4181A on-chip USBFU is connected to the VR4181A internal PCI bus and includes the configuration space. Accessing the USBFU operational registers is enabled by making initial settings to the USBFU configuration registers.

See the description of the PCI_DevOperAccess() function and PciMapping() function in iopciu.c for initialization setting examples of the configuration space of the internal PCI bus.

See **6.14.3 Configuration space** and **13.5 USBFU Configuration Register Set** in **VR4181A Hardware User's Manual (U16049EJ1)** for details of the initial settings of the USBFU configuration registers.

(6) USBFU interrupt mask setting

Set the mask options of USBFU interrupts using the MSYSINT1 register of the VR4181A on-chip interrupt control unit, and USB Interrupt Mask register 1 and USB Interrupt Mask register 2 of the USBFU. Release the masking of interrupts if necessary.

In this sample program, a polling operation is performed by checking the transmit/receive status without using interrupts.

### 6.1.2 VR4181A USBFU internal I/O settings

This section explains the settings of the transmit/receive mailbox, transmit packet descriptor, transmit buffer descriptor, receive packet descriptor, and receive buffer descriptor, that were described in **4.1 VR4181A USBFU Operation**.

(1) Mailbox settings:

usbfunc.c from line 149 to line 175

The VR4181A USBFU has a mailbox function for storing the transmit/receive status in the memory (see p.607 in the **VR4181A User's Manual**). The mailbox is secured in the memory, and is updated each time a data segment is transmitted or received.

In this sample program, the mailbox area is secured in the SDRAM space. Since the length of a receive status is two words, the receive mailbox is secured in 2-word units (Rx_Indication type). Conversely, the length of a transmit status is one word, so the transmit mailbox is secured in 1-word units (Tx_Indication type).

The start address of the secured transmit mailbox area is set to the USB Tx MailBox Start Address register, and the end address to the USB Tx MailBox Bottom Address register. In the same manner, the start address of the secured receive mailbox area is set to the USB Rx MailBox Start Address register, and the end address to the USB Rx MailBox Bottom Address register.

(2) Setting of receive buffer descriptor for specifying receive data buffer:

usbfunc.c from line 180 to line 196

Set the receive buffer descriptor for specifying the receive data buffer for EP0 (see p.613 in the **VR4181A User's Manual**).

First, secure the receive data buffer, and the receive buffer descriptor and receive link pointer for specifying the receive data buffer in the SDRAM using the snow_get_rxbuf() function. The receive buffer descriptor and

receive link pointer are secured in 2-word units, and they configure the receive buffer directory.

Next, set the start address of the receive buffer directory to the USB Command Extension register using the snow_add_buffer() function, and issue the command to add the receive pool to the USB Command register.

(3) Setting of transmit buffer descriptor for specifying transmit data buffer:

usbfunc.c from line 201 to line 225

Set the transmit buffer descriptor for specifying the transmit data buffer for EP0 (see p.611 in the **VR4181A User's Manual**).

### 6.1.3 USB device descriptor initial settings

This section explains the initial settings of device descriptor, configuration descriptor, interface descriptor, endpoint descriptor, and other descriptors described in **3.2 Descriptor**.

(1) Initialization of the device descriptor:

usbfunc.c from line 231 to line 256

Refer to the USB specifications for details of this descriptor.

(2) Initialization of the configuration descriptor, interface descriptor, HID descriptor, and endpoint descriptor:

usbfunc.c from line 258 to line 298

Refer to the USB specifications for details of these descriptors.

This information includes interface information, HID class information, and endpoint information.

(3) Initialization of the string descriptor:

usbfunc.c from line 300 to line 325

Refer to the USB specifications for details of this descriptor.

This information must be prepared in response to a request from the USB host if data exists at offset addresses 14 to 16 (iManufacturer, iProduct, iSerialNumber) of device descriptor information.

(4) Initialization of the HID report descriptor:

usbfunc.c from line 328 to line 380

Refer to the USB specifications for details of this descriptor.

This information must be prepared in response to a request sent for each class after the initial setting. This information indicates the descriptor's relationship with the HID class.

(5) Enabling the use of the Control register at the endpoint:

usbfunc.c from line 391 to line 400

The maximum packet size is set to the USB Endpoint Control register (USB EPn Control register, n = 0 to 6), the EPnEN bit is set to 1 for the used endpoint to enable transmission/reception at endpoint n.

The USB host and USB function uses endpoint 0 for receiving the initial settings, so transmission/reception at endpoint 0 must be enabled beforehand.

In addition, endpoint 5 (interrupt transfer, for IN transaction) is used for transmitting mouse device data after the initial setting is complete, so set the maximum transfer size to the USB EP5 Control register, and set

EP5EN to 1.

(6) Wait state for USB cable connection:

Line 412 of usbfunc.c

The device polls the current state in the snow_isr() function, and waits for cable connection. Specifically, the device polls the read pointer of the transmit/receive mailbox, the write pointer value, and USB General Status register 1, and performs the following depending on each case.

- When the read pointer of the transmit mailbox and the write pointer value differ:
  → Calls the snow_txdone() function and performs transmission.
- When the read pointer of the receive mailbox and the write pointer value differ:
  → Calls the snow_rxdone() function and performs receive end processing.
- When each error bit of USB General Status register 1 is set:
  → Outputs the error details as a debug serial message.

snow_isr() is usually an interrupt servicing function, but a polling operation is used in this sample program.

### 6.1.4 Operation after USB cable connection

When the USB cable is connected, the USB host acquires device information using control transfer. The VR4181A USBFU uses endpoint 0 for control transfer.

The VR4181A USBFU writes the receive status to the receive mailbox, and performs DMA transfer to send the receive data to the receive data buffer. The USBFU device driver reads the receive status and receive data, determines the next processing, and executes it.

(1) USB cable connection

Connect the USB function connector (CN10, Mini-B type) on the Ostrich board to the connector on the USB function side (Mini-B type) of the USB cable.

* When the USB cable is connected, the USB host (PC) uses control transfer to acquire device information. Subsequent operations are performed between the USB host and device via control transfer.

(2) Reception of setup packets from the USB host (EP0 receive status ON)

The VR4181A USBFU transfers the received setup packet to the receive data buffer in the SDRAM, and writes the receive status in the receive mailbox of the SDRAM. Since the data is received at endpoint 0, set 001: Endpoint 0 in the EPN(2:0) field of the receive status. The write pointer of the receive mailbox is then updated. See **Figure 13-18 USBFU Hardware Reception Steps (Normal Mode)** on page 612 in the **VR4181A Hardware User's Manual (U16049EJ1)** for details of the receive operation of the VR4181A USBFU.

(3) Receive data analysis (determined to be the Get_Descriptor command and device descriptor request)

Since the write pointer of the receive mailbox has been updated, the snow_rxdone() function is called in the snow_isr() function (snow_rxdone() function: usbfunc.c from line 1549).

The receive status in the receive mailbox is read using the Snow_rxdone() function. In this sample program, the data is received at endpoint 0, so the snow_rxreq() function is called (line 1614 of usbfunc.c).

The Snow_rxreq() function (usbfunc.c from line 765) analyzes whether the receive data is a setup packet or not (line 777of usbfunc.c). Next, the start address of the receive buffer descriptor is read from the Address(31:0) field of the receive status, and the address of the receive data buffer specified by this descriptor is read.

Next, the receive data is analyzed. In this sample program, the receive data is determined to be the Get_Descriptor command (line 809 of usbfunc.c). The Snow_get_descriptor() function is called and the receive data is determined to be the device descriptor request (line 1147 of usbfunc.c).

(4) Transmission of device descriptor information prepared in **6.1.3 (1)**

Device descriptor information is stored in the transmit data buffer (line 1149 of usbfunc.c), and the snow_tx() function is called to transmit this data using control transfer (line 1222 of usbfunc.c).

(5) Reception of an OUT token from the USB host (EP0 receive status ON)

An OUT token is received from the USB host.

(6) Receive data analysis: SIZE = 0 is confirmed.

(7) Transmission of NULL data from the device to the USB host (line 889 of usbfunc.c)

* The USB host requests the device for a reset based on the device descriptor information.

(8) Transmission of reset signal

The reset signal is sent from the USB host. As a result, the device shifts to the default state (see **3.1.1 State of device**).

★ At this time, the URST bit (bit 17) of USB General Status Register 2 (offset: 0x18) of the VR4181A USBFU is set. A software reset can be applied to the USBFU by writing 1 to bit 15 of the PCIDMC register (offset: 0x04) of the VR4181A USBFU Configuration register. The value of bit 15 is automatically cleared after reset.

PCICMD (0x04) bit 15 warm reset

After software reset execution, re-set the contents described in **6.1.2 VR4181A USBFU internal I/O settings**.

* Since the data in the SDRAM is retained, it is not required to re-set the contents described in **6.1.**3 USB device descriptor initial settings.

* Next the USB host assigns the device-specific address to the device based on the device descriptor information. Device configuration is started after this assignment.

(9) Reception of setup packets from the USB host (EP0 receive status ON)

(10) Receive data analysis (determined to be the Set_Address command)

In the same manner as (3), the receive data sent from the USB host is determined to be the Set_Address command.

(11) Address information setting

Address information in the receive data from the USB host is set in the FA(6:0) field of the USB General Mode register (line 881 of usbfunc.c). As a result, the device shifts to the address state (see **3.1.1 State of device**).

(12) Transmission of NULL data from the device to the USB host (line 889 of usbfunc.c)

* The USB host then recognizes that this specific address is assigned to a USB device. After the Set_Address command is issued, transmission/reception is performed to the assigned specific address.

* The USB host requested device descriptor information (in (2) to (4)), but the USB host only recognized the device descriptor size at that time. To acquire detailed device information, the USB host requests the device descriptor information again.

(13) The same processing as that in (2) to (7).

* The USB host requests device information, and then device configuration information. Based on this information, the USB host acquires information such as the class of the device, the number of used endpoints, and the number of interfaces. The USB host requests the configuration descriptor size when requesting for the first time.

(14) Reception of setup packets from the USB host (EP0 receive status ON)

(15) Receive data analysis (determined to be the Get_Descriptor command and configuration descriptor request)

(16) Transmission of the requested size of configuration descriptor information prepared in **6.1.3 (2)**

Configuration descriptor information is stored in the transmit data buffer (line 1157 of usbfunc.c) and the snow_tx() function is called to transmit this data using control transfer (line 1222 of usbfunc.c).

(17) Reception of an OUT token from the USB host (EP0 receive status ON)

(18) Transmission of NULL data from the device to the USB host

* The USB host acquires the size of the configuration descriptor information and then requests the accurate size of the configuration descriptor.

(19) Reception of setup packets from the USB host (EP0 receive status ON)

(20) Receive data analysis (determined to be the Get_Descriptor command and configuration descriptor request)

(21) Transmission of configuration descriptor information prepared in **6.1.3 (2)**
The same processing as that in (16) is performed.

(22) Reception of an OUT token from the USB host (EP0 receive status ON)

(23) Transmission of NULL data from the device to the USB host

* The USB host requests device descriptor information again. After that, the number of requests and the request contents vary depending on the USB driver in the USB host. The device driver on the device side must be able to respond to any request.

(24) The same processing as that in (2) to (7).

* The USB host requests configuration information on the device side again. At this time, only the descriptor size information is requested.

(25) The same processing as that in (14) to (18).

* The USB host acquires the size of the configuration descriptor information and then requests the accurate size of the configuration descriptor.

(26) The same processing as that in (19) to (23).

* The USB host receives configuration descriptor information, and then transmits the Set_Configuration command.

(27) Reception of setup packets from the USB host (EP0 receive status ON)

(28) Receive data analysis (determined to be the Set_Configuration command)

(29) Transmission of NULL data from the device to the USB host
As a result, the device shifts to the configuration state (see **3.1.1 State of Device**), and users can then use the device.

The USB host requests device descriptor information and configuration information on the device side several times, but this is due to the USB host driver specifications. The device driver for the device side must be created so that the device can respond to these requests.

### 6.1.5 USB-HID class initial settings

Note that the device request issued by the USB host varies depending on the class from this step onward (see **3.1.2 Device requests** for the class commands). The following shows an example of the HID class mouse device.

* After issuing the Set_Configurarion command, the USB host transmits the IDLE request of the class command. Transmit NULL in response to this request.

(1) Reception of setup packets from the USB host (EP0 receive status ON)

(2) Receive data analysis (determined to be the IDLE request of the class command)

(3) Transmission of NULL data from the device to the USB host

* The USB host requests the HID report descriptor information of the device.

(4) Reception of setup packets from the USB host (EP0 receive end status ON)

(5) Receive data analysis (determined to be the Get_Descriptor command and HID report descriptor request)

(6) Transmission of HID report descriptor data prepared in **6.1.3 (4)**

HID report descriptor information is stored in the transmit data buffer (line 1197 of usbfunc.c) and the snow_tx() function is called to transmit this data using control transfer (line 1222 of usbfunc.c).

(7) Reception of an OUT request from the USB host (EP0 receive end status ON)

(8) Transmission of NULL data from the device to the USB host

As a result of the operations up to here, the USB host recognizes that the device is a USB mouse device, and the USB mouse device is recognized in the human interface field of the device manager on the USB host. Communication between the USB host and the device is then established.

### 6.1.6 Sending USB mouse device position data to USB host

This sample program assumes that the VR4181A Ostrich board operates as the USB mouse device. This section explains the procedure for sending the USB mouse device position data to the USB host.

The USB mouse device transmits the current position data to the USB host as 4-byte data. Because interrupt transfer is used for sending the 4-byte data, endpoint 5 of the VR4181A USBFU is specified.

(1) Preparation of the transmit data and transmission to the USB host using EP5.

The transmit data buffer and transmit buffer descriptor are prepared (usbfunc.c from line 421 to 433).

Then, the snow_tx() function is called to transmit the data.

First, the start address of the prepared transmit buffer descriptor is set to the USB Command Extension register (line 1705 of usbfunc.c).

Next, the transmit command is issued to the USB Command register using EP5 (line 1706 of usbfunc.c).

As a result, the mouse cursor on the PC monitor moves in accordance with the transmit data from the USBFU.

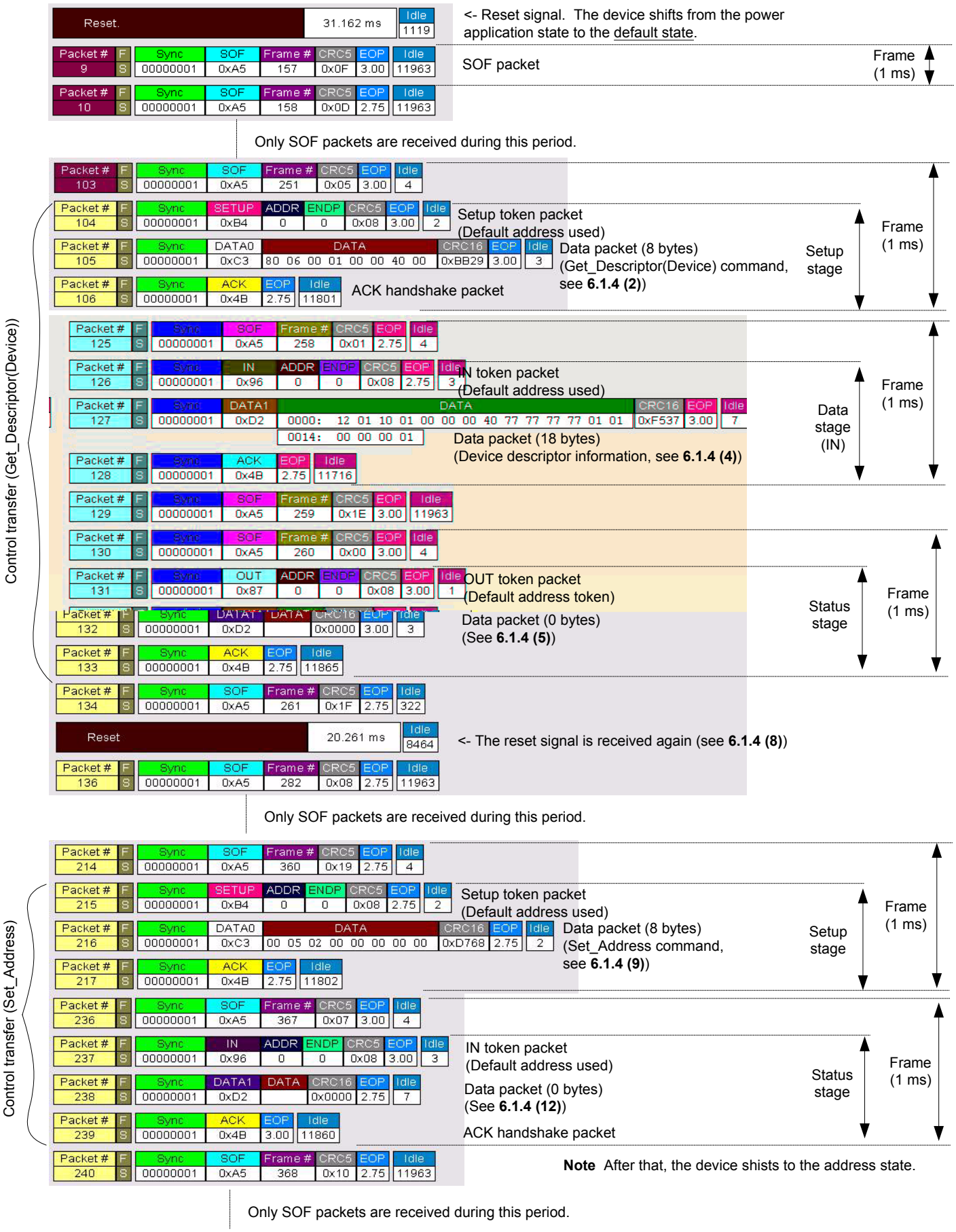
# 7 ANALYZER DATA

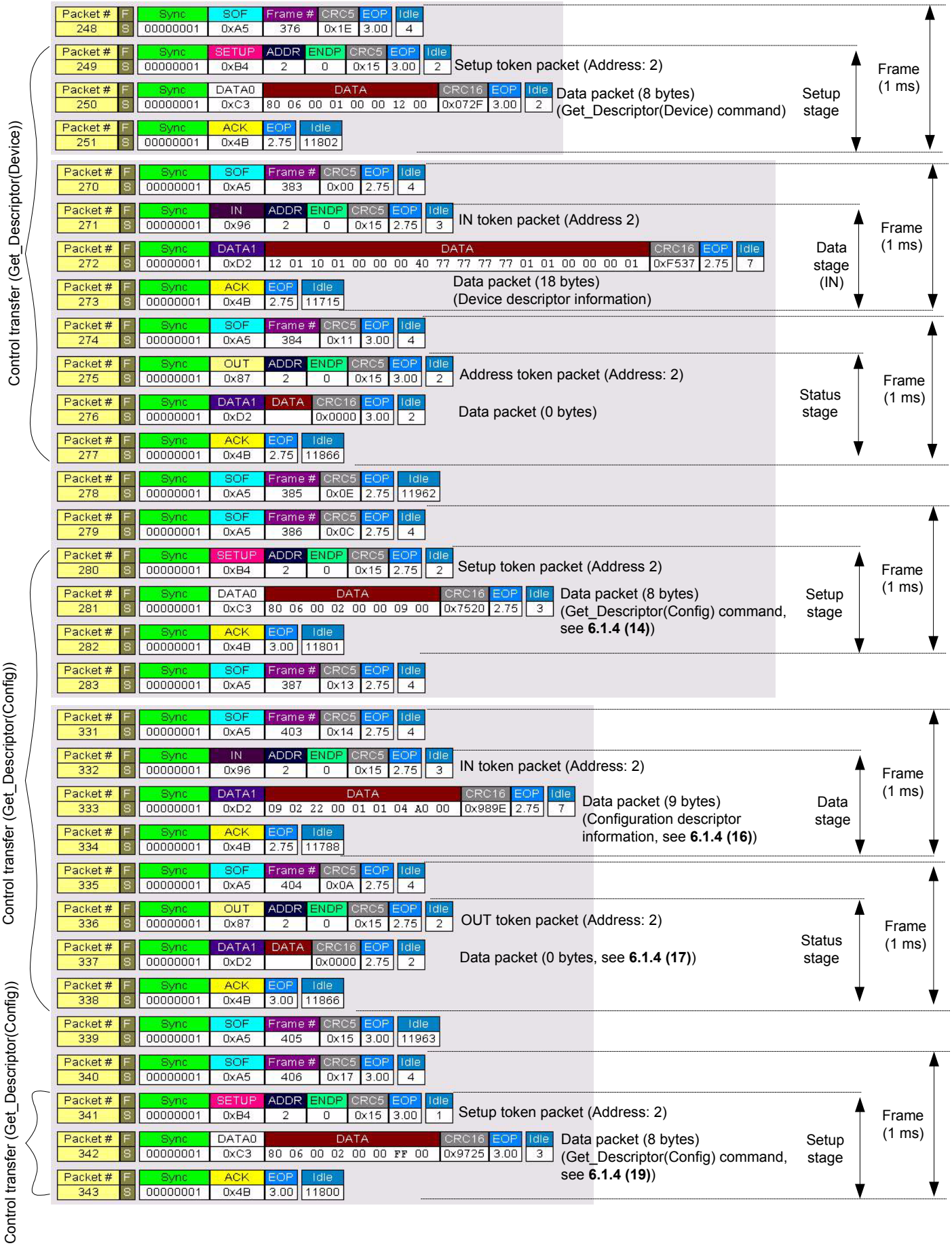
This chapter explains the actual data flow in the bus when measurement is performed using the CATC USB protocol analyzer USB Chief (TOYO Corporation (http://www.toyo.co.jp/)) with the VR4181A on-chip USBFU, by taking the control transfer when a device is connected and interrupt-IN transfer when transferring mouse device data as examples.

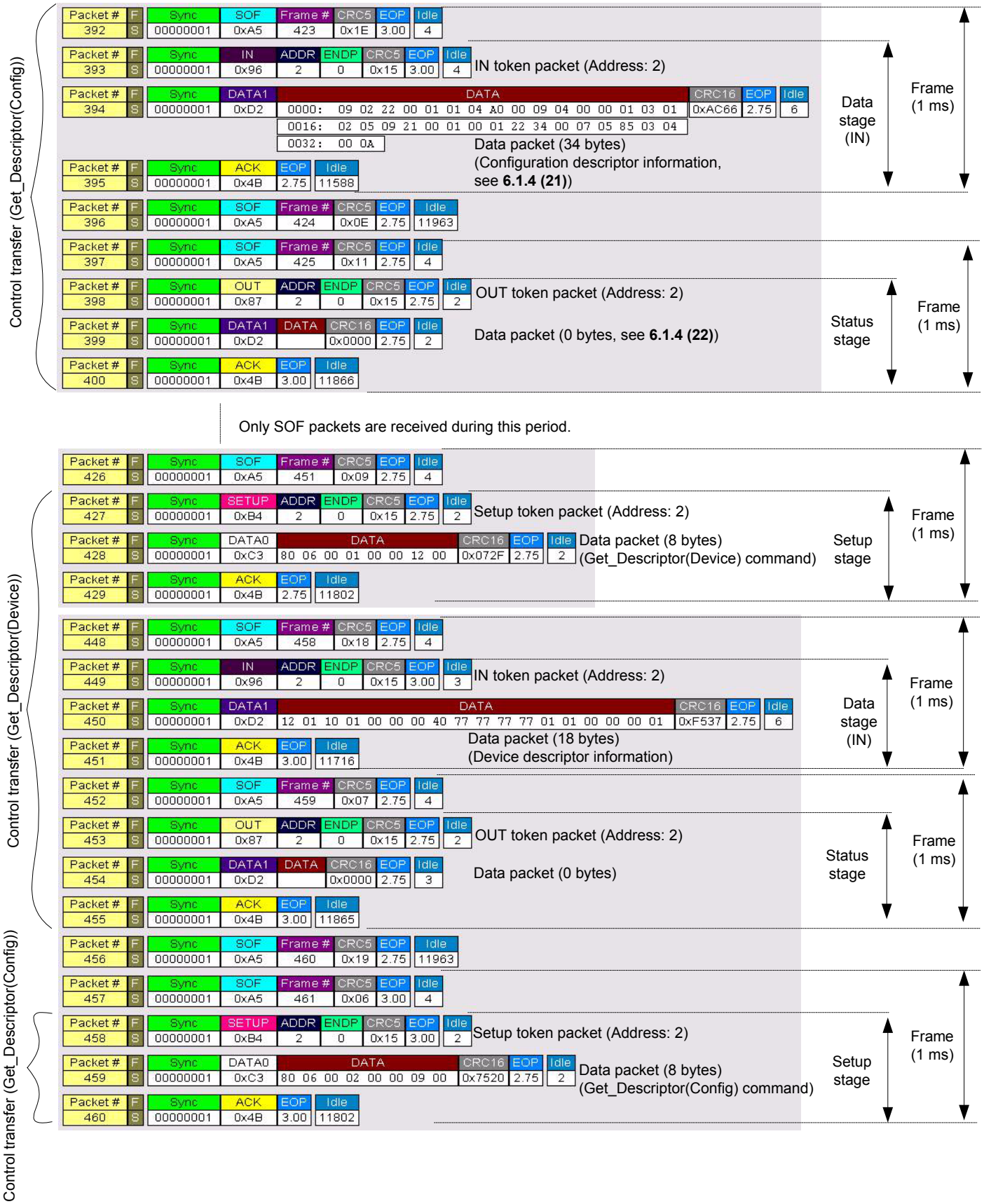
**Note** "Packet#" in the front section of each packet indicates the consecutive number for measurement.
"Idle" in the rear section indicates the idle state between packets.

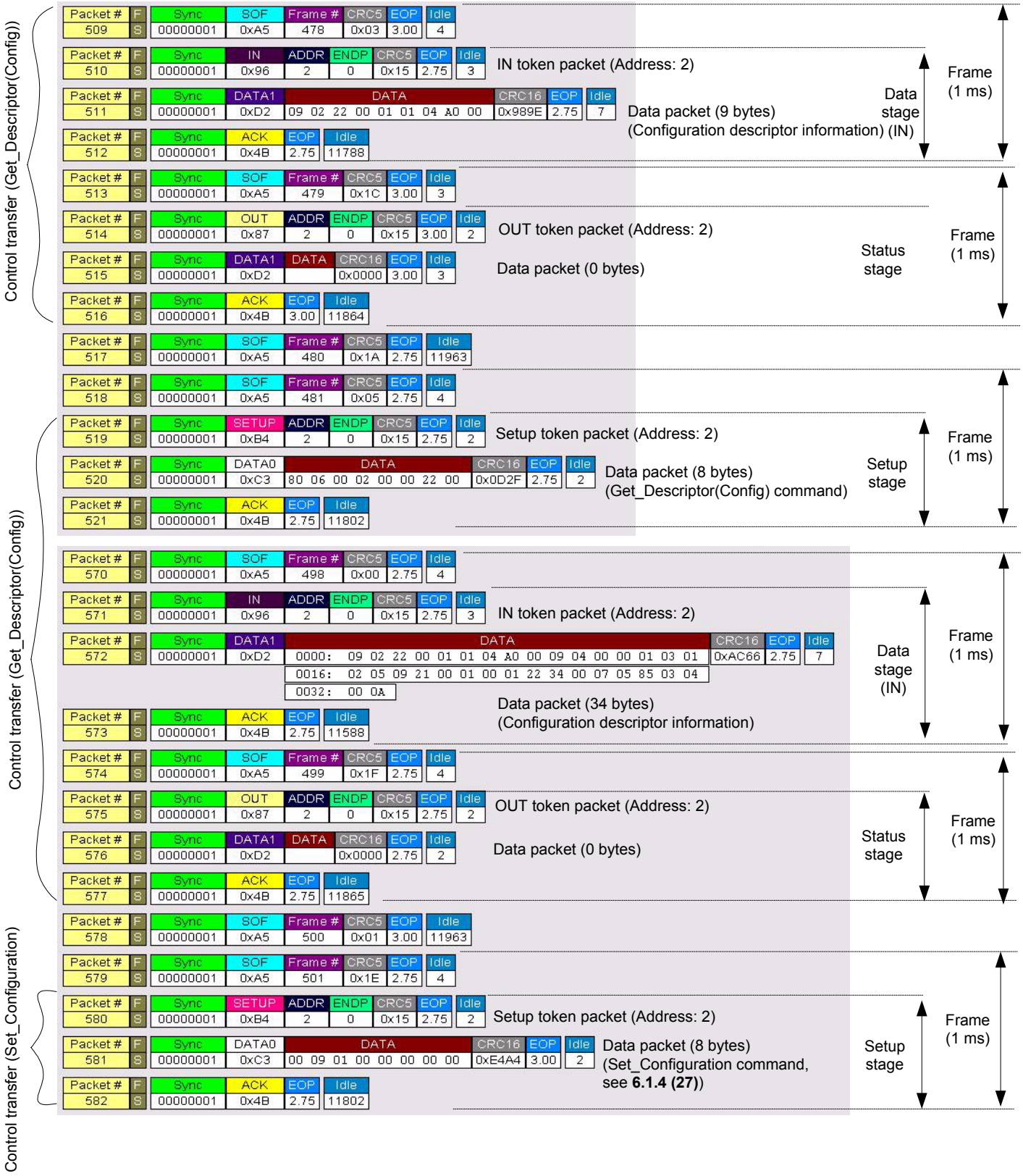
## 7.1 Control Transfer When Device Is Connected

Figure 9 shows the process from when the device is connected to the host controller in the state in which the power is supplied (power application state) to when the device can be used (configuration state).


Reset.
31.162 ms
Idle 1119
<- Reset signal. The device shifts from the power application state to the default state.
Packet # 9 F S | Sync 00000001 | SOF 0xA5 | Frame # 157 | CRC5 0x0F | EOP 3.00 | Idle 11963
SOF packet
Frame (1 ms)
Packet # 10 F S | Sync 00000001 | SOF 0xA5 | Frame # 158 | CRC5 0x0D | EOP 2.75 | Idle 11963
Only SOF packets are received during this period.
Control transfer (Get_Descriptor(Device))
Packet # 103 F S | Sync 00000001 | SOF 0xA5 | Frame # 251 | CRC5 0x05 | EOP 3.00 | Idle 4
Packet # 104 F S | Sync 00000001 | SETUP 0xB4 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 2
Setup token packet (Default address used)
Packet # 105 F S | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 01 00 00 40 00 | CRC16 0xBB29 | EOP 3.00 | Idle 3
Data packet (8 bytes) (Get_Descriptor(Device) command, see 6.1.4 (2))
Packet # 106 F S | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11801
ACK handshake packet
Setup stage
Frame (1 ms)
Packet # 125 F S | Sync 00000001 | SOF 0xA5 | Frame # 258 | CRC5 0x01 | EOP 2.75 | Idle 4
Packet # 126 F S | Sync 00000001 | IN 0x96 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 2.75 | Idle 3
IN token packet (Default address used)
Packet # 127 F S | Sync 00000001 | DATA1 0xD2 | DATA 0000: 12 01 10 01 00 00 00 40 77 77 77 77 01 01 0014: 00 00 00 01 | CRC16 0xF537 | EOP 3.00 | Idle 7
Data packet (18 bytes) (Device descriptor information, see 6.1.4 (4))
Packet # 128 F S | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11716
Data stage (IN)
Frame (1 ms)
Packet # 129 F S | Sync 00000001 | SOF 0xA5 | Frame # 259 | CRC5 0x1E | EOP 3.00 | Idle 11963
Packet # 130 F S | Sync 00000001 | SOF 0xA5 | Frame # 260 | CRC5 0x00 | EOP 3.00 | Idle 4
Packet # 131 F S | Sync 00000001 | OUT 0x87 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 1
OUT token packet (Default address token)
Packet # 132 F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 3
Data packet (0 bytes) (See 6.1.4 (5))
Packet # 133 F S | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11865
Status stage
Frame (1 ms)
Packet # 134 F S | Sync 00000001 | SOF 0xA5 | Frame # 261 | CRC5 0x1F | EOP 2.75 | Idle 322
Reset
20.261 ms
Idle 8464
<- The reset signal is received again (see 6.1.4 (8))
Packet # 136 F S | Sync 00000001 | SOF 0xA5 | Frame # 282 | CRC5 0x08 | EOP 2.75 | Idle 11963
Only SOF packets are received during this period.
Control transfer (Set_Address)
Packet # 214 F S | Sync 00000001 | SOF 0xA5 | Frame # 360 | CRC5 0x19 | EOP 2.75 | Idle 4
Packet # 215 F S | Sync 00000001 | SETUP 0xB4 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 2.75 | Idle 2
Setup token packet (Default address used)
Packet # 216 F S | Sync 00000001 | DATA0 0xC3 | DATA 00 05 02 00 00 00 00 00 | CRC16 0xD768 | EOP 2.75 | Idle 2
Data packet (8 bytes) (Set_Address command, see 6.1.4 (9))
Packet # 217 F S | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11802
Setup stage
Frame (1 ms)
Packet # 236 F S | Sync 00000001 | SOF 0xA5 | Frame # 367 | CRC5 0x07 | EOP 3.00 | Idle 4
Packet # 237 F S | Sync 00000001 | IN 0x96 | ADDR 0 | ENDP 0 | CRC5 0x08 | EOP 3.00 | Idle 3
IN token packet (Default address used)
Packet # 238 F S | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 2.75 | Idle 7
Data packet (0 bytes) (See 6.1.4 (12))
Packet # 239 F S | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11860
ACK handshake packet
Status stage
Frame (1 ms)
Packet # 240 F S | Sync 00000001 | SOF 0xA5 | Frame # 368 | CRC5 0x10 | EOP 2.75 | Idle 11963
Note After that, the device shists to the address state.
Only SOF packets are received during this period.


Control transfer (Get_Descriptor(Device))
Packet # 248 | Sync 00000001 | SOF 0xA5 | Frame # 376 | CRC5 0x1E | EOP 3.00 | Idle 4
Packet # 249 | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Setup token packet (Address: 2)
Packet # 250 | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 01 00 00 12 00 | CRC16 0x072F | EOP 3.00 | Idle 2
Data packet (8 bytes)
(Get_Descriptor(Device) command)
Packet # 251 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11802
Setup stage
Frame (1 ms)
Packet # 270 | Sync 00000001 | SOF 0xA5 | Frame # 383 | CRC5 0x00 | EOP 2.75 | Idle 4
Packet # 271 | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 3
IN token packet (Address 2)
Packet # 272 | Sync 00000001 | DATA1 0xD2 | DATA 12 01 10 01 00 00 00 40 77 77 77 77 01 01 00 00 00 01 | CRC16 0xF537 | EOP 2.75 | Idle 7
Data packet (18 bytes)
(Device descriptor information)
Packet # 273 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11715
Data stage (IN)
Frame (1 ms)
Packet # 274 | Sync 00000001 | SOF 0xA5 | Frame # 384 | CRC5 0x11 | EOP 3.00 | Idle 4
Packet # 275 | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Address token packet (Address: 2)
Packet # 276 | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 2
Data packet (0 bytes)
Packet # 277 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11866
Status stage
Frame (1 ms)
Packet # 278 | Sync 00000001 | SOF 0xA5 | Frame # 385 | CRC5 0x0E | EOP 2.75 | Idle 11962
Packet # 279 | Sync 00000001 | SOF 0xA5 | Frame # 386 | CRC5 0x0C | EOP 2.75 | Idle 4
Control transfer (Get_Descriptor(Config))
Packet # 280 | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
Setup token packet (Address 2)
Packet # 281 | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 02 00 00 09 00 | CRC16 0x7520 | EOP 2.75 | Idle 3
Data packet (8 bytes)
(Get_Descriptor(Config) command, see 6.1.4 (14))
Packet # 282 | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11801
Setup stage
Frame (1 ms)
Packet # 283 | Sync 00000001 | SOF 0xA5 | Frame # 387 | CRC5 0x13 | EOP 2.75 | Idle 4
Packet # 331 | Sync 00000001 | SOF 0xA5 | Frame # 403 | CRC5 0x14 | EOP 2.75 | Idle 4
Packet # 332 | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 3
IN token packet (Address: 2)
Packet # 333 | Sync 00000001 | DATA1 0xD2 | DATA 09 02 22 00 01 01 04 A0 00 | CRC16 0x989E | EOP 2.75 | Idle 7
Data packet (9 bytes)
(Configuration descriptor information, see 6.1.4 (16))
Packet # 334 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11788
Data stage
Frame (1 ms)
Packet # 335 | Sync 00000001 | SOF 0xA5 | Frame # 404 | CRC5 0x0A | EOP 2.75 | Idle 4
Packet # 336 | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
OUT token packet (Address: 2)
Packet # 337 | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 2.75 | Idle 2
Data packet (0 bytes, see 6.1.4 (17))
Packet # 338 | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11866
Status stage
Frame (1 ms)
Packet # 339 | Sync 00000001 | SOF 0xA5 | Frame # 405 | CRC5 0x15 | EOP 3.00 | Idle 11963
Packet # 340 | Sync 00000001 | SOF 0xA5 | Frame # 406 | CRC5 0x17 | EOP 3.00 | Idle 4
Control transfer (Get_Descriptor(Config))
Packet # 341 | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 1
Setup token packet (Address: 2)
Packet # 342 | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 02 00 00 FF 00 | CRC16 0x9725 | EOP 3.00 | Idle 3
Data packet (8 bytes)
(Get_Descriptor(Config) command, see 6.1.4 (19))
Packet # 343 | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11800
Setup stage
Frame (1 ms)


Control transfer (Get_Descriptor(Config))
Packet # 392 | SOF | Frame # 423 | CRC5 0x1E | EOP 3.00 | Idle 4
Packet # 393 | IN | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 4
IN token packet (Address: 2)
Packet # 394 | DATA1 | DATA 0000: 09 02 22 00 01 01 04 A0 00 09 04 00 00 01 03 01 0016: 02 05 09 21 00 01 00 01 22 34 00 07 05 85 03 04 0032: 00 0A | CRC16 0xAC66 | EOP 2.75 | Idle 6
Data packet (34 bytes)
(Configuration descriptor information, see 6.1.4 (21))
Packet # 395 | ACK | EOP 2.75 | Idle 11588
Packet # 396 | SOF | Frame # 424 | CRC5 0x0E | EOP 2.75 | Idle 11963
Packet # 397 | SOF | Frame # 425 | CRC5 0x11 | EOP 2.75 | Idle 4
Packet # 398 | OUT | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
OUT token packet (Address: 2)
Packet # 399 | DATA1 | DATA | CRC16 0x0000 | EOP 2.75 | Idle 2
Data packet (0 bytes, see 6.1.4 (22))
Packet # 400 | ACK | EOP 3.00 | Idle 11866
Data stage (IN)
Status stage
Frame (1 ms)
Frame (1 ms)
Only SOF packets are received during this period.
Control transfer (Get_Descriptor(Device))
Packet # 426 | SOF | Frame # 451 | CRC5 0x09 | EOP 2.75 | Idle 4
Packet # 427 | SETUP | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
Setup token packet (Address: 2)
Packet # 428 | DATA0 | DATA 80 06 00 01 00 00 12 00 | CRC16 0x072F | EOP 2.75 | Idle 2
Data packet (8 bytes)
(Get_Descriptor(Device) command)
Packet # 429 | ACK | EOP 2.75 | Idle 11802
Packet # 448 | SOF | Frame # 458 | CRC5 0x18 | EOP 2.75 | Idle 4
Packet # 449 | IN | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 3
IN token packet (Address: 2)
Packet # 450 | DATA1 | DATA 12 01 10 01 00 00 00 40 77 77 77 77 01 01 00 00 00 01 | CRC16 0xF537 | EOP 2.75 | Idle 6
Data packet (18 bytes)
(Device descriptor information)
Packet # 451 | ACK | EOP 3.00 | Idle 11716
Packet # 452 | SOF | Frame # 459 | CRC5 0x07 | EOP 2.75 | Idle 4
Packet # 453 | OUT | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
OUT token packet (Address: 2)
Packet # 454 | DATA1 | DATA | CRC16 0x0000 | EOP 2.75 | Idle 3
Data packet (0 bytes)
Packet # 455 | ACK | EOP 3.00 | Idle 11865
Data stage (IN)
Status stage
Frame (1 ms)
Frame (1 ms)
Control transfer (Get_Descriptor(Config))
Packet # 456 | SOF | Frame # 460 | CRC5 0x19 | EOP 2.75 | Idle 11963
Packet # 457 | SOF | Frame # 461 | CRC5 0x06 | EOP 3.00 | Idle 4
Packet # 458 | SETUP | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
Setup token packet (Address: 2)
Packet # 459 | DATA0 | DATA 80 06 00 02 00 00 09 00 | CRC16 0x7520 | EOP 2.75 | Idle 2
Data packet (8 bytes)
(Get_Descriptor(Config) command)
Packet # 460 | ACK | EOP 3.00 | Idle 11802
Setup stage
Frame (1 ms)
Frame (1 ms)


Control transfer (Get_Descriptor(Config))
Packet # 509 | Sync 00000001 | SOF 0xA5 | Frame # 478 | CRC5 0x03 | EOP 3.00 | Idle 4
Packet # 510 | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 3
IN token packet (Address: 2)
Packet # 511 | Sync 00000001 | DATA1 0xD2 | DATA 09 02 22 00 01 01 04 A0 00 | CRC16 0x989E | EOP 2.75 | Idle 7
Data packet (9 bytes)
(Configuration descriptor information) (IN)
Packet # 512 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11788
Data stage
Frame (1 ms)
Packet # 513 | Sync 00000001 | SOF 0xA5 | Frame # 479 | CRC5 0x1C | EOP 3.00 | Idle 3
Packet # 514 | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 3.00 | Idle 2
OUT token packet (Address: 2)
Packet # 515 | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 3.00 | Idle 3
Data packet (0 bytes)
Status stage
Packet # 516 | Sync 00000001 | ACK 0x4B | EOP 3.00 | Idle 11864
Frame (1 ms)
Packet # 517 | Sync 00000001 | SOF 0xA5 | Frame # 480 | CRC5 0x1A | EOP 2.75 | Idle 11963
Packet # 518 | Sync 00000001 | SOF 0xA5 | Frame # 481 | CRC5 0x05 | EOP 2.75 | Idle 4
Control transfer (Get_Descriptor(Config))
Packet # 519 | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
Setup token packet (Address: 2)
Packet # 520 | Sync 00000001 | DATA0 0xC3 | DATA 80 06 00 02 00 00 22 00 | CRC16 0x0D2F | EOP 2.75 | Idle 2
Data packet (8 bytes)
(Get_Descriptor(Config) command)
Packet # 521 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11802
Setup stage
Frame (1 ms)
Packet # 570 | Sync 00000001 | SOF 0xA5 | Frame # 498 | CRC5 0x00 | EOP 2.75 | Idle 4
Packet # 571 | Sync 00000001 | IN 0x96 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 3
IN token packet (Address: 2)
Packet # 572 | Sync 00000001 | DATA1 0xD2 | DATA 0000: 09 02 22 00 01 01 04 A0 00 09 04 00 00 01 03 01 | 0016: 02 05 09 21 00 01 00 01 22 34 00 07 05 85 03 04 | 0032: 00 0A | CRC16 0xAC66 | EOP 2.75 | Idle 7
Data packet (34 bytes)
(Configuration descriptor information)
Packet # 573 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11588
Data stage (IN)
Frame (1 ms)
Packet # 574 | Sync 00000001 | SOF 0xA5 | Frame # 499 | CRC5 0x1F | EOP 2.75 | Idle 4
Packet # 575 | Sync 00000001 | OUT 0x87 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
OUT token packet (Address: 2)
Packet # 576 | Sync 00000001 | DATA1 0xD2 | DATA | CRC16 0x0000 | EOP 2.75 | Idle 2
Data packet (0 bytes)
Status stage
Packet # 577 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11865
Frame (1 ms)
Packet # 578 | Sync 00000001 | SOF 0xA5 | Frame # 500 | CRC5 0x01 | EOP 3.00 | Idle 11963
Packet # 579 | Sync 00000001 | SOF 0xA5 | Frame # 501 | CRC5 0x1E | EOP 2.75 | Idle 4
Control transfer (Set_Configuration)
Packet # 580 | Sync 00000001 | SETUP 0xB4 | ADDR 2 | ENDP 0 | CRC5 0x15 | EOP 2.75 | Idle 2
Setup token packet (Address: 2)
Packet # 581 | Sync 00000001 | DATA0 0xC3 | DATA 00 09 01 00 00 00 00 00 | CRC16 0xE4A4 | EOP 3.00 | Idle 2
Data packet (8 bytes)
(Set_Configuration command,
see 6.1.4 (27))
Packet # 582 | Sync 00000001 | ACK 0x4B | EOP 2.75 | Idle 11802
Setup stage
Frame (1 ms)

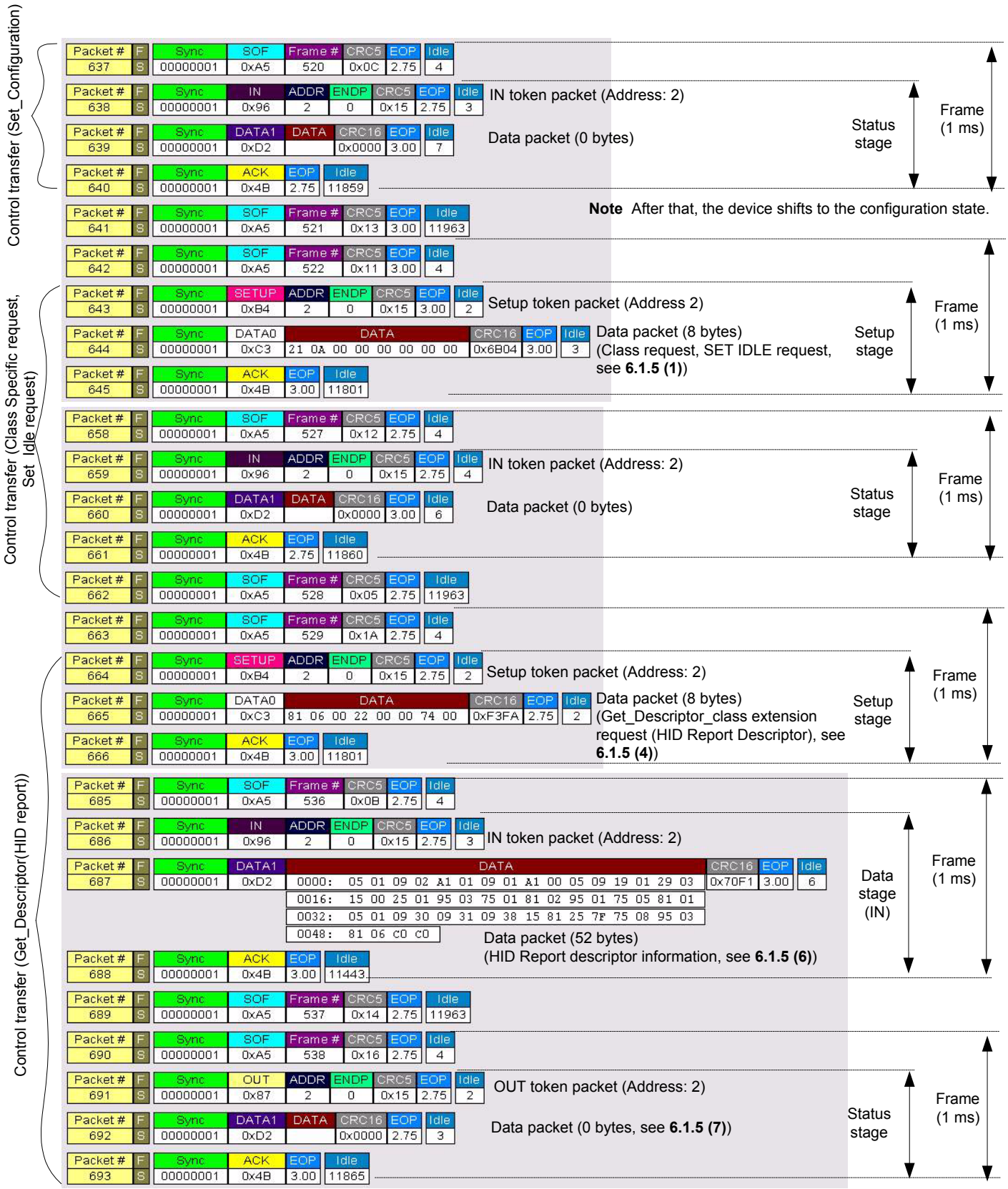



**Figure 9. Control Transfer When Device Is Connected**

## 7.2 Interrupt-IN Transfer to Transfer Mouse Device Data

Figure 10 shows the measurement result when the mouse device data is transferred from the device to the host controller using an interrupt-IN transfer.

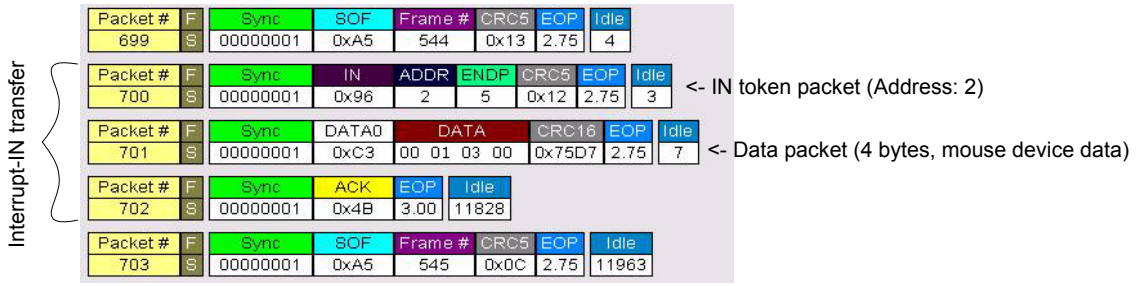



**Figure 10. Interrupt-IN Transfer to Transfer Mouse Device Data**

# APPENDIX A. SAMPLE PROGRAM SOURCE CODE

The sample program (the Usbfunc.c file) described in this document is shown below.

```
1 :     /*
2 :      *          USBF sample program
3 :      *          Copyright (C) NMS-RYOSAN Ltd. 2002
4 :      *
5 :      */
6 :     /*****************************************************************************/
7 :     /* INCLUDE FILES                                                             */
8 :     /*****************************************************************************/
9 :     #include <stdlib.h>
10 :
11 :    #include "usbfunc.h"
12 :    #include "iopci.h"
13 :    #include "icu_vr4181a.h"
14 :    #include "ccu_vr4181a.h"
15 :
16 :    /*****************************************************************************/
17 :    /* MACRO DEFINITION                                                          */
18 :    /*****************************************************************************/
19 :
20 :    /*****************************************************************************/
21 :    /* EXTERNAL VARIABLES                                                        */
22 :    /*****************************************************************************/
23 :
24 :    /*****************************************************************************/
25 :    /* EXTERNAL FUNCTIONS                                                        */
26 :    /*****************************************************************************/
27 :
28 :    /*****************************************************************************/
29 :    /* GLOBAL VARIABLES                                                          */
30 :    /*****************************************************************************/
31 :    /**** Pointer to last linkpointer *****/
32 :    Buffer_Desc * sn_BottomOfList[3];
33 :
34 :    /**** Secure the area for Mailbox ****/
35 :    #define FOR_EXTRA_32BYTE_8          8
36 :    #define FOR_EXTRA_32BYTE_4          4
37 :    Tx_Indication   sn_ForTxMailBox[ SN_TX_MLBNUM + FOR_EXTRA_32BYTE_8 ], *sn_TxMailBox;
38 :    Rx_Indication   sn_ForRxMailBox[ SN_RX_MLBNUM + FOR_EXTRA_32BYTE_4 ], *sn_RxMailBox;
39 :
40 :    /***** Flag that indicates transfer mode *****/
41 :    /*int sn_TxModeFlag = BULK_TXMODE;*/
42 :    int sn_TxModeFlag = ( INT_RXMODE | ISO_RXMODE | BULK_RXMODE );
43 :    /*int sn_RxModeFlag = ( BULK_RXMODE );*/
44 :    int sn_RxModeFlag =  ( INT_RXMODE | ISO_RXMODE | BULK_RXMODE );
45 :
46 :    /*****For Tx Data*******/
47 :    #define                                 TX_BDESC_SIZE                   4
48 :    #define                                 TX_BUFFER_SIZE                  128
49 :    Buffer_Desc             sn_ForTxBufferDesc[ TX_BDESC_SIZE ], *sn_TxBufferDesc;
50 :    unsigned char   sn_ForTxDataBuffer[ TX_BUFFER_SIZE ], *sn_TxDataBuffer;
51 :
52 :    /*****For Tx 0-Length Data*******/
53 :    Buffer_Desc             *sn_TxBufferDesc_Length0;
54 :    unsigned char   *sn_TxDataBuffer_Length0;
55 :
56 :    /*****Device Request relation structure definition.********/
57 :    De_Desc                 sn_DeDesc;
58 :    Co_Desc                 sn_CoDesc;
59 :    St_Desc                 sn_StDesc;
60 :    StIn_Desc       sn_StIndexDesc;
61 :    Re_Desc                 sn_ReDesc;
62 :    Status                  sn_Device_Status;
63 :    Dev_Req                 sn_DevReq;
64 :    Value                   sn_Value;
65 :
66 :    /******Device Request relation global variable definition.*******/
67 :    int                                             sn_wait_data_flag = 0;
68 :    int                                             sn_set_addressflag = 0;
69 :    unsigned char   sn_ep0size;
70 :    int                                     sn_setup_flag = 0;
71 :
72 :    /*****************************************************************************/
73 :    /* PROTOTYPES                                                                */
74 :    /*****************************************************************************/
75 :    void USBF_Init( void );
76 :    Buffer_Desc* snow_get_rxbuf( unsigned long PoolNum );
77 :    void snow_add_buffer( Buffer_Desc * p_desc_top );
78 :    void AddBuffer( int PoolNumber, int NumOfDir, unsigned long Address );
79 :    void snow_isr( void );
80 :
81 :    void snow_rxreq( Rx_Indication * );
82 :    void snow_get_status( unsigned char reqtype, unsigned short index, unsigned short length );
83 :    void snow_clear_feature( unsigned char reqtype, unsigned short value, unsigned short index );
84 :    void snow_set_feature( unsigned char reqtype, unsigned short value, unsigned short index );
85 :    void snow_get_descriptor( unsigned short value, unsigned short length);
86 :    void snow_set_descriptor( unsigned short value, unsigned short length, unsigned long address );
87 :    void snow_get_configuration( unsigned short length );
88 :    void snow_set_configuration( unsigned char value );
89 :    void snow_get_interface( unsigned short length );
90 :    void snow_set_interface( unsigned char value );
91 :    void snow_synch_frame( void );
92 :
93 :    void snow_rxdone( void );
94 :    int             snow_tx( int, unsigned long, unsigned long );
95 :    void snow_txdone( void );
```

```
 96 :
 97 :          /*************************************************************************/
 98 :          /* FUNCTION DEFINITION                                                   */
 99 :          /*************************************************************************/
100 :          /*********************************************************/
101 :          /* USBF_Init                                             */
102 :          /*********************************************************/
103 :          void            USBF_Init( void )
104 :          {
105 :                          int                                     i;
106 :                          Buffer_Desc * AddressOfTopDir;
107 :                          unsigned long tmp;
108 :                          unsigned long addr;
109 :                          //-----------------------------------------------------
110 :                          //              InitUSB
111 :                          //-----------------------------------------------------
112 :                          RS_puts( "Start USBF Setup!\r\n" );
113 :
114 :                          //-----------------------------------------------------
115 :                          //              GPIO Init USBF
116 :                          //-----------------------------------------------------
117 :
118 :                          // USB 48MHz Clock Input Enable
119 :          //              outph( PINMODED2, ( 0x4000 | ( inph( PINMODED2 ) & 0x3FFF ) ) );
120 :                          // USB Buffer Enable
121 :          //              outph( USBSIGCTRL, ( 0x000f | ( inph( USBSIGCTRL ) & 0xFFF0 ) ) );
122 :
123 :                          //-----------------------------------------------------
124 :                          //              USBF CLOCK SET
125 :                          //-----------------------------------------------------
126 :
127 :                          // USB 48MHz Clock Input Enable
128 :          //              outph( CMUCLKMSK1, ( inph( CMUCLKMSK1 ) | 0x3010 ) );     // CLK48 for USBF,USBH and BITCLK for AC97
129 :                          // PCI CMUCLKMSK OPEN
130 :          //              outpw( CMUCLKMSK3, ( inpw( CMUCLKMSK3 ) | 0x003a ) );    // Supply PCIClock to AC97,USBF,USBH. Supply TClock and PCIClock to
PCI bridge (IOPCIU).
131 :
132 :                          //-----------------------------------------------------
133 :                          //              USBF INTERRUPT SET
134 :                          //-----------------------------------------------------
135 :
136 :                          // USBF SYSINT1 Setting
137 :          //              outph( MSYSINT1REG, ( inph( MSYSINT1REG ) | 0x2000 ) );                                 // Enable USBF interrupt notification.
138 :                          // USBF interrupt mask . No interrupt set.
139 :          //              outpw( USBF_INTMSK1, ( inpw( USBF_INTMSK1 ) | 0x80000003 ) );           // Mask all USBF interrupt
140 :                          // USBF interrupt mask . No interrupt set.
141 :          //              outpw( USBF_INTMSK2, ( inpw( USBF_INTMSK2 ) | 0x00070000 ) );           // Mask all USBF interrupt
142 :
143 :                          //-----------------------------------------------------
144 :                          //              USBF MAILBOX or POOL initilize
145 :                          //-----------------------------------------------------
146 :                          //-----------------------------------------------------
147 :                          //              Alining the pointer to beginning of Mailbox with cacheline
148 :                          //-----------------------------------------------------
149 :                          tmp = (unsigned long) &sn_ForTxMailBox[0];
150 :                          tmp = ( tmp + 0x10 ) & 0xfffffff0;                                                      // Alining
151 :                          sn_TxMailBox = (Tx_Indication *)tmp;
152 :
153 :                          tmp = (unsigned long) &sn_ForRxMailBox[0];
154 :                          tmp = ( tmp + 0x10 ) & 0xfffffff0;                                                      // Alining
155 :                          sn_RxMailBox = (Rx_Indication *)tmp;
156 :
157 :                          //-----------------------------------------------------
158 :                          //              Initializing the Mailbox. All 0.
159 :                          //-----------------------------------------------------
160 :                          for( i=0; i<SN_TX_MLBNUM; i++ )                 {
161 :                                          sn_TxMailBox[i].Word[0] = 0x0;
162 :                          }
163 :                          for( i=0; i<SN_RX_MLBNUM; i++ ) {
164 :                                          sn_RxMailBox[i].Word[0] = 0x0;
165 :                                          sn_RxMailBox[i].Word[1] = 0x0;
166 :                          }
167 :
168 :                          //-----------------------------------------------------
169 :                          //              Whiting the Address of Mailbox to Register
170 :                          //-----------------------------------------------------
171 :                          outpw( USBF_TxMail_START_ADDR, CPUtoPCI( (unsigned long)&sn_TxMailBox[0] ) );
172 :                          outpw( USBF_TxMail_BTM_ADDR, CPUtoPCI( (unsigned long)&sn_TxMailBox[ SN_TX_MLBNUM ] ) );
173 :
174 :                          outpw( USBF_RxMail_START_ADDR, CPUtoPCI( (unsigned long)&sn_RxMailBox[0] ) );
175 :                          outpw( USBF_RxMail_BTM_ADDR, CPUtoPCI( (unsigned long)&sn_RxMailBox[ SN_RX_MLBNUM ] ) );
176 :
177 :                          //-----------------------------------------------------
178 :                          //              Preparing the Rx Buffer
179 :                          //-----------------------------------------------------
180 :                          /* Pool0 */
181 :                          AddressOfTopDir = snow_get_rxbuf( 0 );
182 :                          snow_add_buffer( AddressOfTopDir );
183 :
184 :                          /* Pool1 */
185 :                          if( sn_RxModeFlag & ISO_RXMODE )
186 :                          {
187 :                                          AddressOfTopDir = snow_get_rxbuf( 1 );
188 :                                          snow_add_buffer( AddressOfTopDir );
189 :                          }
190 :
191 :                          /* Pool2 */
192 :                          if( sn_RxModeFlag & BULK_RXMODE )
193 :                          {
194 :                                          AddressOfTopDir = snow_get_rxbuf( 2 );
195 :                                          snow_add_buffer( AddressOfTopDir );
196 :                          }
197 :
198 :                          //-----------------------------------------------------
199 :                          // Initiarize the Descriptor for Tx
```

```
200 :                       //-----------------------------------------------------
201 :                       tmp = ( unsigned long ) &sn_ForTxBufferDesc[0];
202 :                       tmp = ( tmp + 0x10 ) & 0xfffffff0;
203 :                       sn_TxBufferDesc = ( Buffer_Desc * )tmp;
204 :
205 :                       sn_TxBufferDesc[0].Word[0] = 0x0;
206 :
207 :                       tmp = ( unsigned long ) &sn_ForTxDataBuffer[0];
208 :                       tmp = ( tmp + 0x10 ) & 0xfffffff0;
209 :                       sn_TxDataBuffer = ( unsigned char * )tmp;
210 :
211 :                       //-----------------------------------------------------
212 :                       // Initiarize the Descriptor for Tx (0-Length Packet)
213 :                       //-----------------------------------------------------
214 :                       sn_TxBufferDesc_Length0 =  &sn_TxBufferDesc[2];
215 :                       sn_TxDataBuffer_Length0 =  &sn_TxDataBuffer[2];
216 :
217 :                       /* prepare Buffer Directory */
218 :                       sn_TxBufferDesc_Length0[0].Word[0] = ( unsigned long )( LASTBIT | DLBIT  );
219 :                       sn_TxBufferDesc_Length0[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer_Length0 );
220 :
221 :                       sn_TxBufferDesc_Length0[1].Word[0] = 0;
222 :                       sn_TxBufferDesc_Length0[1].Word[1] = 0;
223 :
224 :                       /* prepare Payload Data to SDRAM */
225 :                       sn_TxDataBuffer_Length0[0] = 0x0;
226 :
227 :                       /*-------------------------------------------------------------------------*/
228 :                       /* Initiarize the Descriptor for ( Device, Other )                                                  */
229 :                       /*-------------------------------------------------------------------------*/
230 :
231 :                       // DEVICE_DESCRIPTOR ////////////////////////////////////////////////////
232 :                       sn_DeDesc.Desc[0] = 0x12;      /* bLength         */
233 :                       sn_DeDesc.Desc[1] = 0x1;       /* bDescriptorType */
234 :                       sn_DeDesc.Desc[2] = 0x10;      /* bcdUSB          */
235 :                       sn_DeDesc.Desc[3] = 0x01;
236 :                       sn_DeDesc.Desc[4] = 0x00;      /* bDeviceClass    */
237 :                       sn_DeDesc.Desc[5] = 0x00;      /* bDeviceSubClass */
238 :                       sn_DeDesc.Desc[6] = 0x00;      /* bDeviceProtocol */
239 :
240 :                       sn_DeDesc.Desc[7] = ( unsigned char )MAXP0;/* bMaxPacketSize0 */
241 :                       sn_DeDesc.Desc[8] = 0x77;      /* for the time being *//* idVendor */
242 :                       sn_DeDesc.Desc[9] = 0x77;      /* for the time being */
243 :                       sn_DeDesc.Desc[10] = 0x77;     /* for the time being *//* idProduct */
244 :                       sn_DeDesc.Desc[11] = 0x77;     /* for the time being */
245 :                       sn_DeDesc.Desc[12] = 0x01;     /* bcdDevice          */
246 :                       sn_DeDesc.Desc[13] = 0x01;
247 :       #if 0           // Put 0 and disable STRING INDEX if STRING INDEX is not used.
248 :                       sn_DeDesc.Desc[14] = 0x00;     /* iManufacture 00    */
249 :                       sn_DeDesc.Desc[15] = 0x00;     /* iProduct 00        */
250 :                       sn_DeDesc.Desc[16] = 0x00;     /* iSerialNumber      */
251 :       #else           // Use the following value if STRING INDEX is used.
252 :                       sn_DeDesc.Desc[14] = 0x01;
253 :                       sn_DeDesc.Desc[15] = 0x02;
254 :                       sn_DeDesc.Desc[16] = 0x05;
255 :       #endif
256 :                       sn_DeDesc.Desc[17] = 0x01;     /* bNumConfigurations */
257 :
258 :                       // CONFIGRATION_DESCRIPTOR //////////////////////////////////////////////
259 :                       sn_CoDesc.Desc[0] = 0x09;      /* bLength             */
260 :                       sn_CoDesc.Desc[1] = 0x02;      /* bDescriptorType     */
261 :                       sn_CoDesc.Desc[2] = 0x22;      /* wTotalLength        */
262 :                       sn_CoDesc.Desc[3] = 0x00;
263 :                       sn_CoDesc.Desc[4] = 0x01;      /* bNumInterfaces      */
264 :                       sn_CoDesc.Desc[5] = 0x01;      /* bConfigurationValue */
265 :                       sn_CoDesc.Desc[6] = 0x04;      /* iConfiguration      */
266 :                       sn_CoDesc.Desc[7] = 0xa0;      /* bmAttributes        */
267 :                       sn_CoDesc.Desc[8] = 0x00;      /* MaxPower            */
268 :
269 :                       // Interface0 Descriptor
270 :                       sn_CoDesc.Desc[9] = 0x09;      /* bLength               */
271 :                       sn_CoDesc.Desc[10] = 0x04;     /* bDescriptorType       */
272 :                       sn_CoDesc.Desc[11] = 0x00;     /* bInterfaceNumber      */
273 :                       sn_CoDesc.Desc[12] = 0x00;     /* bAltemateSetting      */
274 :                       sn_CoDesc.Desc[13] = 0x01;     /* bNumEndpoints         */
275 :                       sn_CoDesc.Desc[14] = 0x03;     /* bInterfaceClass    0xff */
276 :                       sn_CoDesc.Desc[15] = 0x01;     /* bInterfaceSubClass 0xff */
277 :                       sn_CoDesc.Desc[16] = 0x02;     /* bInterfaceProtocol 0x00 */
278 :                       sn_CoDesc.Desc[17] = 0x05;     /* iInterface            */
279 :
280 :                       // HID Descriptor
281 :                       sn_CoDesc.Desc[18] = 0x09;     /* bLength                                          */
282 :                       sn_CoDesc.Desc[19] = 0x21;     /* bDescriptorType                                  */
283 :                       sn_CoDesc.Desc[20] = 0x00;     /* bcdHID                                           */
284 :                       sn_CoDesc.Desc[21] = 0x01;
285 :                       sn_CoDesc.Desc[22] = 0x00;     /* bCountryCode                                     */
286 :                       sn_CoDesc.Desc[23] = 0x01;     /* bNumDescriptors                                  */
287 :                       sn_CoDesc.Desc[24] = 0x22;     /* bDescriptorType                                  */
288 :                       sn_CoDesc.Desc[25] = 0x34;     /* wDescriptorlength      */
289 :                       sn_CoDesc.Desc[26] = 0x00;
290 :
291 :                       // Endpoint Descriptor
292 :                       sn_CoDesc.Desc[27] = 0x7;      /* bLength                                          */
293 :                       sn_CoDesc.Desc[28] = 0x5;      /* bDescriptorType       */
294 :                       sn_CoDesc.Desc[29] = 0x85;     /* bEndpointAddress      */
295 :                       sn_CoDesc.Desc[30] = 0x03;     /* bmAttributes                                     */
296 :                       sn_CoDesc.Desc[31] = 0x04;     /* wMaxPacketSize        */
297 :                       sn_CoDesc.Desc[32] = 0x00;
298 :                       sn_CoDesc.Desc[33] = 0xa;      /* bInterval                                        */
299 :
300 :                       // STRING_DESCRIPTOR ////////////////////////////////////////////////////
301 :                       // String Descriptor
302 :                       sn_StDesc.Desc[0] = 0x04;
303 :                       sn_StDesc.Desc[1] = 0x03;
304 :                       sn_StDesc.Desc[2] = 0x09;
```

```
305 :                   sn_StDesc.Desc[3] = 0x04;
306 :
307 :                   // STRING_INDEX_DESCRIPTOR ////////////////////////////////////////
308 :                   sn_StIndexDesc.Desc[0] = 0x12;                  /* bLength          */
309 :                   sn_StIndexDesc.Desc[1] = 0x03;
310 :                   sn_StIndexDesc.Desc[2] = 0x55;                  // U
311 :                   sn_StIndexDesc.Desc[3] = 0x00;
312 :                   sn_StIndexDesc.Desc[4] = 0x53;                  // S
313 :                   sn_StIndexDesc.Desc[5] = 0x00;
314 :                   sn_StIndexDesc.Desc[6] = 0x42;                  // B
315 :                   sn_StIndexDesc.Desc[7] = 0x00;
316 :                   sn_StIndexDesc.Desc[8] = 0x54;                  // T
317 :                   sn_StIndexDesc.Desc[9] = 0x00;
318 :                   sn_StIndexDesc.Desc[10] = 0x45;                 // E
319 :                   sn_StIndexDesc.Desc[11] = 0x00;
320 :                   sn_StIndexDesc.Desc[12] = 0x53;                 // S
321 :                   sn_StIndexDesc.Desc[13] = 0x00;
322 :                   sn_StIndexDesc.Desc[14] = 0x54;                 // T
323 :                   sn_StIndexDesc.Desc[15] = 0x00;
324 :                   sn_StIndexDesc.Desc[16] = 0x21;                 // !
325 :                   sn_StIndexDesc.Desc[17] = 0x00;
326 :
327 :                   // HID REPORT DESCRIPTOR ////////////////////////////////////////////
328 :                   // Report Descriptor
329 :                   sn_ReDesc.Desc[0] = 0x05;     /* Usage Page (Generic Desktop Control) */
330 :                   sn_ReDesc.Desc[1] = 0x01;
331 :                   sn_ReDesc.Desc[2] = 0x09;     /* Usage (Mouse)                                    */
332 :                   sn_ReDesc.Desc[3] = 0x02;
333 :                   sn_ReDesc.Desc[4] = 0xA1;     /* Collection (Application)                 */
334 :                   sn_ReDesc.Desc[5] = 0x01;
335 :                   sn_ReDesc.Desc[6] = 0x09;     /* Usage (Pointer)                                                  */
336 :                   sn_ReDesc.Desc[7] = 0x01;
337 :                   sn_ReDesc.Desc[8] = 0xA1;     /* Collection (Physical)                            */
338 :                   sn_ReDesc.Desc[9] = 0x00;
339 :                   sn_ReDesc.Desc[10] = 0x05;    /* Usage Page (Button)                                              */
340 :                   sn_ReDesc.Desc[11] = 0x09;
341 :                   sn_ReDesc.Desc[12] = 0x19;    /* Usage Minimum (1)                                                */
342 :                   sn_ReDesc.Desc[13] = 0x01;
343 :                   sn_ReDesc.Desc[14] = 0x29;    /* Usage Maximum (3)                                                */
344 :                   sn_ReDesc.Desc[15] = 0x03;
345 :                   sn_ReDesc.Desc[16] = 0x15;    /* Logical Minimum (0)                                              */
346 :                   sn_ReDesc.Desc[17] = 0x00;
347 :                   sn_ReDesc.Desc[18] = 0x25;    /* Logical Maximum (1)                                              */
348 :                   sn_ReDesc.Desc[19] = 0x01;
349 :                   sn_ReDesc.Desc[20] = 0x95;    /* Report Count (3)                                                 */
350 :                   sn_ReDesc.Desc[21] = 0x03;
351 :                   sn_ReDesc.Desc[22] = 0x75;    /* Report Size (1)                                                  */
352 :                   sn_ReDesc.Desc[23] = 0x01;
353 :                   sn_ReDesc.Desc[24] = 0x81;    /* Input (Data, Variable, Absolute)                 */
354 :                   sn_ReDesc.Desc[25] = 0x02;
355 :                   sn_ReDesc.Desc[26] = 0x95;    /* Report Count (1)                                                 */
356 :                   sn_ReDesc.Desc[27] = 0x01;
357 :                   sn_ReDesc.Desc[28] = 0x75;    /* Report Size (5)                                  */
358 :                   sn_ReDesc.Desc[29] = 0x05;
359 :                   sn_ReDesc.Desc[30] = 0x81;    /* Input (Constant)                                                 */
360 :                   sn_ReDesc.Desc[31] = 0x01;
361 :                   sn_ReDesc.Desc[32] = 0x05;    /* Usage Page (Generic Desktop Control) */
362 :                   sn_ReDesc.Desc[33] = 0x01;
363 :                   sn_ReDesc.Desc[34] = 0x09;    /* Usage (X)                                                        */
364 :                   sn_ReDesc.Desc[35] = 0x30;
365 :                   sn_ReDesc.Desc[36] = 0x09;    /* Usage (Y)                                                        */
366 :                   sn_ReDesc.Desc[37] = 0x31;
367 :                   sn_ReDesc.Desc[38] = 0x09;    /* Usage (Wheel)                                                    */
368 :                   sn_ReDesc.Desc[39] = 0x38;
369 :                   sn_ReDesc.Desc[40] = 0x15;    /* Logical Minimum (-127)                           */
370 :                   sn_ReDesc.Desc[41] = 0x81;
371 :                   sn_ReDesc.Desc[42] = 0x25;    /* Logical Maximum (127)                            */
372 :                   sn_ReDesc.Desc[43] = 0x7F;
373 :                   sn_ReDesc.Desc[44] = 0x75;    /* Report Size (8)                                                  */
374 :                   sn_ReDesc.Desc[45] = 0x08;
375 :                   sn_ReDesc.Desc[46] = 0x95;    /* Report Count (3)                                                 */
376 :                   sn_ReDesc.Desc[47] = 0x03;
377 :                   sn_ReDesc.Desc[48] = 0x81;    /* Input (Data, Variable, Relative)                 */
378 :                   sn_ReDesc.Desc[49] = 0x06;
379 :                   sn_ReDesc.Desc[50] = 0xC0;    /* End Collection                                                   */
380 :                   sn_ReDesc.Desc[51] = 0xC0;    /* End Collection                                                   */
381 :
382 :                   //  ////////////////////////////////////////////////////////////////
383 :                   sn_Value.Number[0] = sn_CoDesc.Desc[5];
384 :                   sn_Value.Number[1] = sn_CoDesc.Desc[12];
385 :                   sn_ep0size  = ( unsigned char )MAXP0;
386 :
387 :                   //  ////////////////////////////////////////////////////////////////
388 :                   sn_Device_Status.Status[1] |= SELF_POWERED;
389 :
390 :                   //-----------------------------------------------------
391 :                   //                  Setting the Endpoint Controll Register.
392 :                   //                  After setting EP0EN, Devicerequest operation starts.
393 :                   //-----------------------------------------------------
394 :                   outpw( USBF_EP0_CONT, ( ( EP0EN ) | ( MAXP0 ) ) );
395 :                   outpw( USBF_EP1_CONT, ( MAXP1 ) );
396 :                   outpw( USBF_EP2_CONT, ( RM2_NOR | MAXP2 ) );
397 :                   outpw( USBF_EP3_CONT, ( MAXP3 ) );
398 :                   outpw( USBF_EP4_CONT, ( RM4_ASM | MAXP4 ) );
399 :                   outpw( USBF_EP5_CONT, ( ( EP5EN ) | ( MAXP5 ) ) );
400 :                   outpw( USBF_EP2_CONT, ( MAXP6 ) );
401 :
402 :                   //-----------------------------------------------------
403 :                   //                  STATUS CHEACK( USBH • RESET REQUEST )
404 :                   //-----------------------------------------------------
405 :                   RS_puts( "USBF_START_EP0!\r\n" );
406 :
407 :                   while (1)
408 :                   {
409 :                                   //-----------------------------------------------------
```

```
410 :                                   //              STATUS CHEACK( USBH • SETUP REQUEST )
411 :                                   //-----------------------------------------------------
412 :                                   snow_isr( );
413 :
414 :                                   if ( sn_setup_flag == 1 ) {
415 :                                           //-----------------------------------------------------
416 :                                           //              USBF SETUP OK
417 :                                           //-----------------------------------------------------
418 :                                           //              MOUSE DATA TX COMMOND( USBF • HOST-TX )
419 :                                           //-----------------------------------------------------
420 :                                           /* prepare Buffer Directory */
421 :                                           sn_TxBufferDesc[0].Word[0] = ( unsigned long )( LASTBIT | DLBIT | 4 );
422 :                                           sn_TxBufferDesc[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer );
423 :
424 :                                           sn_TxBufferDesc[1].Word[0] = 0;
425 :                                           sn_TxBufferDesc[1].Word[1] = 0;
426 :
427 :                                           /* prepare Payload Data to SDRAM */
428 :                                           sn_TxDataBuffer[0] = 0x00;
429 :                                           sn_TxDataBuffer[1] = 0x01;
430 :                                           sn_TxDataBuffer[2] = 0x03;
431 :                                           sn_TxDataBuffer[3] = 0x0;
432 :
433 :                                           snow_tx( INT_TXMODE, 4, ( unsigned long )sn_TxBufferDesc );
434 :                                   }
435 :                           }
436 :
437 :                           RS_puts( "USBF_STOP!\r\n" );
438 :
439 :                           return;
440 :           }
441 :
442 :           /*-------------------------------------------------------------------------*/
443 :           /* argument:                    PoolNum - Pool Number
444 :           /* return value:  Pointer to beggining of BufferDirectory
445 :           /* use:                                                    Getting the RxBuffer
446 :           /* contents:                    Getting the RxBufferDirectory.
447 :           /*                                                                 Getting the RxBuffer.
448 :           /*                                                                 Setting the RxBufferDescriptor.
449 :           /*-------------------------------------------------------------------------*/
450 :           Buffer_Desc* snow_get_rxbuf( unsigned long PoolNum )
451 :           {
452 :                           int                             i, j;
453 :                           int             DirNumInPool;
454 :                           Buffer_Desc * p_RxDesc, *p_top;
455 :                           unsigned long PoolNumOfDesc;
456 :                           unsigned char * p_RxBuff;
457 :                           unsigned long tmp;
458 :
459 :                           RS_puts( "snow_get_rxbuf\r\n" );
460 :
461 :                           switch( PoolNum )
462 :                           {
463 :                                   case 0:
464 :                                           DirNumInPool = SN_DIRNUM_POOL0;
465 :                                           PoolNumOfDesc = POOLNUM_0;
466 :                                           break;
467 :                                   case 1:
468 :                                           DirNumInPool = SN_DIRNUM_POOL1;
469 :                                           PoolNumOfDesc = POOLNUM_1;
470 :                                           break;
471 :                                   case 2:
472 :                                           DirNumInPool = SN_DIRNUM_POOL2;
473 :                                           PoolNumOfDesc = POOLNUM_2;
474 :                                           break;
475 :                                   default:
476 :                                           break;
477 :                           }
478 :
479 :                           /* Securing the area for BufferDirectory. And storing the beginning address */
480 :                           tmp = ( unsigned long ) malloc( ( ( ( SN_DESCNUM_IN_DIR + 1 ) * DirNumInPool ) * sizeof( Buffer_Desc ) ) + EXTRA_32BYTE );
481 :                           tmp = ( tmp + 0x10 ) & 0xfffffff0;
482 :                           p_RxDesc = ( Buffer_Desc * )tmp;
483 :
484 :                           p_top = p_RxDesc;
485 :
486 :                           /* Securing the area for DataBuffer */
487 :                           tmp = ( unsigned long ) malloc( ( ( SN_DESCNUM_IN_DIR * DirNumInPool ) * SN_RX_BUFSIZE ) + EXTRA_32BYTE );
488 :                           tmp = ( tmp + 0x10 ) & 0xfffffff0;
489 :                           p_RxBuff = ( unsigned char * )tmp;
490 :
491 :                           /* Setting the BufferDescriptor by ones*/
492 :                           for( j=0; j < DirNumInPool ; j++ )
493 :                           {
494 :                                   for(i=0; i < SN_DESCNUM_IN_DIR ; i++)
495 :                                   {
496 :                                           p_RxDesc->Word[0] = ( DLBIT | PoolNumOfDesc | SN_RX_BUFSIZE );
497 :                                           p_RxDesc->Word[1] = CPUtoPCI( ( unsigned long )p_RxBuff );
498 :                                           p_RxBuff += SN_RX_BUFSIZE;
499 :
500 :                                           p_RxDesc++;
501 :                                   }
502 :
503 :                                   p_RxDesc->Word[0] = PoolNumOfDesc;
504 :                                   p_RxDesc->Word[1] = CPUtoPCI( ( unsigned long )( p_RxDesc + 1 ) );
505 :
506 :                                   p_RxDesc++;
507 :                           }
508 :
509 :                           /* Back one. And let linkpointer to point NULL */
510 :                           p_RxDesc--;
511 :                           p_RxDesc->Word[1] = 0x0;
512 :                           /* Storing the pointer to last linkpointer for Add_Buffer operation. */
513 :                           sn_BottomOfList[PoolNum] = p_RxDesc;
```

```
514 :
515 :                   /* Back one. And set the LastBit. */
516 :                   p_RxDesc--;
517 :                   p_RxDesc->Word[0] = ( ( p_RxDesc->Word[0] ) | LASTBIT );
518 :
519 :                   return p_top;
520 :   }
521 :
522 :
523 :   /*-----------------------------------------------------------------------*/
524 :   /*-----------------------------------------------------------------------*/
525 :   /* Add_Buffer Section                                                    */
526 :   /*-----------------------------------------------------------------------*/
527 :   /*-----------------------------------------------------------------------*/
528 :
529 :   /*-----------------------------------------------------------------------*/
530 :   /* argument:                                                             */
531 :   /*                   p_desc_top - top address of Rxbufdirectory          */
532 :   /* return value:  non                                                    */
533 :   /* use:                                  Adding the BufferDirectory to the Pool */
534 :   /* contents:         Counting the number of directories.                  */
535 :   /*                                       Issueing the AddBuffer command. */
536 :   /*-----------------------------------------------------------------------*/
537 :   void snow_add_buffer( Buffer_Desc * p_desc_top )
538 :   {
539 :                   Buffer_Desc * p_desc;
540 :                   Buffer_Desc * NextBottomOfList;
541 :                   unsigned long Command;
542 :                   unsigned long DirNum, LastBit, DLBit;
543 :                   int PoolNum;
544 :                   int LastDir;
545 :
546 :                   RS_puts( "snow_add_buffer\r\n" );
547 :
548 :                   p_desc = p_desc_top;
549 :
550 :                   /*-------------------------------------------------------*/
551 :                   /* Finding PoolNumber. And deciding the command to be issued. */
552 :                   /*-------------------------------------------------------*/
553 :                   if( p_desc->Word[0] & POOLNUM_0 ) {
554 :                           Command = CMD_ADD_POOL0;
555 :                           PoolNum = 0;
556 :                   } else if ( p_desc->Word[0] & POOLNUM_1 ) {
557 :                           Command = CMD_ADD_POOL1;
558 :                           PoolNum = 1;
559 :                   } else if ( p_desc->Word[0] & POOLNUM_2 ) {
560 :                           Command = CMD_ADD_POOL2;
561 :                           PoolNum = 2;
562 :                   }
563 :
564 :                   /*-------------------------------------------------------*/
565 :                   /* Counting the number of BufferDirectory.               */
566 :                   /* And storing the address of last lonkpointer of new directory.  */
567 :                   /*-------------------------------------------------------*/
568 :                   LastDir = 0;
569 :                   DirNum = 0;
570 :                   while ( 1 )
571 :                   {
572 :                           /*---------------------------------------------*/
573 :                           /* Checking the LastBit and DL_Bit.            */
574 :                           /*---------------------------------------------*/
575 :                           if ( p_desc->Word[0] & LASTBIT ) {
576 :                                   LastBit = 1;
577 :                           } else {
578 :                                   LastBit = 0;
579 :                           }
580 :                           if ( p_desc->Word[0] & DLBIT ) {
581 :                                   DLBit = 1;
582 :                           } else {
583 :                                   DLBit = 0;
584 :                           }
585 :
586 :                           /*---------------------------------------------*/
587 :                           /* Checking whether it's BufferDescriptor of LinkPointer. */
588 :                           /*---------------------------------------------*/
589 :                           /* If BufferDescriptor.                        */
590 :                           if( LastBit == 0 && DLBit == 1 )  /* BufferDiscriptor */
591 :                           {
592 :                                   p_desc++;
593 :                           }
594 :                           /* If LinkPointer.                             */
595 :                           else if( LastBit == 0 && DLBit == 0 )   /* LinkPointer */
596 :                           {
597 :                                   DirNum++;
598 :                                   if( LastDir == 1 )
599 :                                   {
600 :                                           NextBottomOfList = p_desc;
601 :                                           break;
602 :                                   }
603 :
604 :                                   p_desc = ( Buffer_Desc * )PCItoCPU( ( unsigned long)( p_desc->Word[1] ) );
605 :                           }
606 :                           /* If it's last BufferDescriptor, Off the LastBit. */
607 :                           else if( LastBit == 1 && DLBit == 1 )   /* Last BufferDiscriptor */
608 :                           {
609 :                                   p_desc->Word[0] &= ( ~LASTBIT );
610 :                                   LastDir = 1;
611 :                                   p_desc++;
612 :                           }
613 :                           else
614 :                           {
615 :                                   break;
616 :                           }
617 :                   }
618 :
```

```
619 :                   /* Baning the interrupt from here.                                                              */
620 :   //              intLevel = intLock();
621 :
622 :                   /*-------------------------------------------------------------------*/
623 :                   /* Wrighting the address of beggining of new directory                                          */
624 :                   /* into Last LinkPointer.                                                                       */
625 :                   /*-------------------------------------------------------------------*/
626 :                   sn_BottomOfList[PoolNum]->Word[1] = CPUtoPCI( ( unsigned long)p_desc_top );
627 :
628 :                   /*-------------------------------------------------------------------*/
629 :                   /* Issueing the command.                                                                        */
630 :                   /*-------------------------------------------------------------------*/
631 :                   while( 1 ){
632 :                           if( ( inpw( USBF_CMD ) & CMR_BUSY ) == 0 ) {
633 :                                   break;
634 :                           }
635 :                   }
636 :
637 :                   outpw( USBF_CMD_ADDRESS, CPUtoPCI( (unsigned long)p_desc_top ) );
638 :                   outpw( USBF_CMD, ( inpw( USBF_CMD ) | ( Command ) | ( DirNum ) ) );
639 :
640 :                   /*-------------------------------------------------------------------*/
641 :                   /* Renewal the pointer to the last linkpinter.                                                  */
642 :                   /*-------------------------------------------------------------------*/
643 :                   sn_BottomOfList[PoolNum] = NextBottomOfList;
644 :
645 :                   /* Baning the interrupt to here.
646 :   //              intUnlock( intLevel );
647 :                   return;
648 :   }
649 :
650 :
651 :   /*-------------------------------------------------------------------*/
652 :   /*-------------------------------------------------------------------*/
653 :   /* Interrupt  Section                                                                                           */
654 :   /*-------------------------------------------------------------------*/
655 :   /*-------------------------------------------------------------------*/
656 :
657 :   /*-------------------------------------------------------------------*/
658 :   /* argument:                 non
659 :   /* return value:  non
660 :   /* use:                                                    Interrupt Service Routine
661 :   /* contents:                 Checking the GSR1.
662 :   /*                                                                   Going to Tx or Rx or ERROR operation.
663 :   /*-------------------------------------------------------------------*/
664 :   void            snow_isr( void )
665 :   {
666 :                   unsigned long gmr,gsr1,gsr2,resetstat;
667 :                   Tx_Indication *TxRead_p, *TxWrite_p;
668 :                   Rx_Indication *RxRead_p, *RxWrite_p;
669 :   //              int intLevel;
670 :
671 :                   /* Baning the interrupt from here. */
672 :   //              intLevel = intLock();
673 :
674 :                   /* Get the value of General Status Register. */
675 :                   gsr1 = inpw( USBF_GSTAT1 );
676 :
677 :                   gmr = inpw( USBF_GMODE );
678 :
679 :                   /* Get the value of Tx/Rx Mailbox Read/Wright Address Register.                                 */
680 :                   TxRead_p  = (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_RD_ADDR ) );
681 :                   TxWrite_p = (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_WR_ADDR ) );
682 :                   RxRead_p  = (Rx_Indication *)PCItoCPU( inpw( USBF_RxMail_RD_ADDR ) );
683 :                   RxWrite_p = (Rx_Indication *)PCItoCPU( inpw( USBF_RxMail_WR_ADDR ) );
684 :
685 :                   /*-------------------------------------------------------------------*/
686 :                   /* Going on while GSR1 is set or while readpointer and wrightpointer                */
687 :                   /* don't point same address.                                                                    */
688 :                   /*-------------------------------------------------------------------*/
689 :
690 :                   while( ( gsr1 & ( TX_FINISH | RX_FINISH | GSR1_ERROR ) )
691 :                                   || TxRead_p != TxWrite_p
692 :                                   || RxRead_p != RxWrite_p )
693 :                   {
694 :                           /* Tx operation                                                                         */
695 :                           if( TxRead_p != TxWrite_p )     {
696 :                                   RS_puts( "TX_OPE\r\n" );
697 :                                   snow_txdone( );
698 :                           }
699 :                           /* Rx operation                                                                         */
700 :                           if( RxRead_p != RxWrite_p )     {
701 :                                   RS_puts( "RX_OPE\r\n" );
702 :                                   snow_rxdone( );
703 :                           }
704 :                           /* Error operation of GSR1.                                                             */
705 :                           if( gsr1 & ( GSR1_ERROR & ~GSR2 ) )             {
706 :                                   if( gsr1 & TMF )RS_puts("TMF\n");
707 :                                   if( gsr1 & RMF )RS_puts("RMF\n");
708 :                                   if( gsr1 & RPE2 )               RS_puts("RPE2\n");
709 :                                   if( gsr1 & RPE1 )               RS_puts("RPE1\n");
710 :                                   if( gsr1 & RPE0 )               RS_puts("RPE0\n");
711 :                                   if( gsr1 & RPA2 )               RS_puts("RPA2\n");
712 :                                   if( gsr1 & RPA1 )               RS_puts("RPA1\n");
713 :                                   if( gsr1 & RPA0 )               RS_puts("RPA0\n");
714 :                                   if( gsr1 & DER )RS_puts("DER\n");
715 :                                   if( gsr1 & EP2FO )              RS_puts("EP2FO\n");
716 :                                   if( gsr1 & EP1FU )              RS_puts("EP1FU\n");
717 :                           }
718 :                           /* Error operation of GSR2.                                                             */
719 :                           if( gsr1 & GSR2 ) {
720 :                                   gsr2 = inpw( USBF_GSTAT2 );
721 :                                   if( gsr2 & ( GSR2_ERROR & ~( FW | URSM | USPD ) ) )             {
722 :                                           if(gsr2 & IFN)                  RS_puts("IFN\n");
723 :                                           if(gsr2 & IEA)                  RS_puts("IEA\n");
```

```
724 :                                                   if(gsr2 & EP2OS)        RS_puts("EP2OS\n");
725 :                                                   if(gsr2 & EP2ED)        RS_puts("EP2ED\n");
726 :                                                   if(gsr2 & EP2ND)        RS_puts("EP2ND\n");
727 :                                                   if(gsr2 & EP1NT)        RS_puts("EP1NT\n");
728 :                                                   if(gsr2 & EP1ET)        RS_puts("EP1ET\n");
729 :                                                   if(gsr2 & EP1ND)        RS_puts("EP1ND\n");
730 :                                                   if(gsr2 & ES)           RS_puts("ES\n");
731 :                                                   if(gsr2 & SL)           RS_puts("SL\n");
732 :                                                   if(gsr2 & URST) {
733 :                                                           RS_puts("RESET\n");
734 :                                                           USBF_Init();
735 :                                                   }
736 :                                           }
737 :                                   }
738 :
739 :                                   gsr1 = inpw( USBF_GSTAT1 );
740 :                                   TxRead_p  = (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_RD_ADDR ) );
741 :                                   TxWrite_p = (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_WR_ADDR ) );
742 :                                   RxRead_p  = (Rx_Indication *)PCItoCPU( inpw( USBF_RxMail_RD_ADDR ) );
743 :                                   RxWrite_p = (Rx_Indication *)PCItoCPU( inpw( USBF_RxMail_WR_ADDR ) );
744 :
745 :                   }
746 :
747 :                   /* Baning the interrupt to here. */
748 :   //              intUnlock( intLevel );
749 :
750 :   }
751 :
752 :
753 :   /*-----------------------------------------------------------------------*/
754 :   /*-----------------------------------------------------------------------*/
755 :   /* Device Request Section                                                                                                */
756 :   /*-----------------------------------------------------------------------*/
757 :   /*-----------------------------------------------------------------------*/
758 :
759 :   /*---------------------------------------------------------*/
760 :   /* argument:                indadd - Rx Indication Address                 */
761 :   /* return value:  non                                                                                                    */
762 :   /* use:                                                    The analysis of Device Request.                               */
763 :   /* contents:                stert by Rx at EndPoint 0.                                    */
764 :   /*---------------------------------------------------------*/
765 :   void             snow_rxreq( Rx_Indication *indadd )
766 :   {
767 :                    unsigned char   reqtype, request, set_value;
768 :                    unsigned short  value, index, length;
769 :                    unsigned char   *RxDataBuffer;
770 :                    unsigned long   address, size;
771 :                    Buffer_Desc             *RxBufferDesc;
772 :
773 :                    RS_puts( "snow_rxreq\r\n" );
774 :
775 :                    RxBufferDesc = (Buffer_Desc *)PCItoCPU( indadd->Word[1] );
776 :
777 :                    if( indadd->Word[0] & ( USB_SETUP_PACKET ) ) {
778 :                                    if( ( ( RxBufferDesc->Word[0] ) & 0xffff ) != REQUEST_SIZE ){
779 :                                      RS_puts( "rxreq size error\r\n" );
780 :                                      return; /* Error */
781 :                                    }
782 :                                    RxDataBuffer = ( unsigned char *)PCItoCPU( RxBufferDesc->Word[1] );
783 :
784 :                                    sn_DevReq = *(Dev_Req *)RxDataBuffer;
785 :
786 :                                    reqtype = sn_DevReq.Request[0];
787 :                                    request = *(unsigned char *)(RxDataBuffer + 1);
788 :                                    value =           *(unsigned short *)(& RxDataBuffer[2]);
789 :                                    index =           *(unsigned short *)(& RxDataBuffer[4]);
790 :                                    length = *(unsigned short *)(& RxDataBuffer[6]);
791 :
792 :                                    set_value =  sn_DevReq.Request[2];
793 :
794 :                                    switch( request ){
795 :                                                    case            GET_STATUS:
796 :                                                                    snow_get_status( reqtype, index, length );
797 :                                                                    break;
798 :                                                    case            CLEAR_FEATURE:
799 :                                                                    snow_clear_feature( reqtype, value, index );
800 :                                                                    break;
801 :                                                    case            SET_FEATURE:
802 :                                                                    snow_set_feature( reqtype, value, index );
803 :                                                                    break;
804 :                                                    case            SET_ADDRESS:
805 :   /*                                                               snow_set_address( value ); */
806 :                                                                    /* setting of Function Address to USB_GMODE */
807 :                                                                    sn_set_addressflag = 1;
808 :                                                                    break;
809 :                                                    case            GET_DESCRIPTOR:
810 :                                                                    snow_get_descriptor( value, length );
811 :                                                                    break;
812 :                                                    case            SET_DESCRIPTOR:
813 :   /*                                                               snow_set_descriptor( value, length, address ); */
814 :                                                                    /* data waiting */
815 :                                                                    sn_wait_data_flag = 1;
816 :                                                                    break;
817 :                                                    case            GET_CONFIGURATION:
818 :                                                                    snow_get_configuration( length );
819 :                                                                    break;
820 :                                                    case            SET_CONFIGURATION:
821 :                                                                    /* Configration value setting */
822 :                                                                    snow_set_configuration( set_value );
823 :                                                                    break;
824 :                                                    case            GET_INTERFACE:
825 :                                                                    snow_get_interface( length );
826 :                                                                    break;
827 :                                                    case            SET_INTERFACE:
828 :                                                                    /* Altemate value setting */
```

```
829 :                                                   snow_set_interface( set_value );
830 :                                                   break;
831 :                                           case    SYNCH_FRAME:
832 :                                                   snow_synch_frame();
833 :                                                   break;
834 :                                           default:    break;
835 :                                   }
836 :                           } else {
837 :                                   /* EndPoint 0 RX */
838 :                                   RS_puts( "EP0 RX\r\n" );
839 :                                   if( ( sn_wait_data_flag == 0 ) && ( ( sn_DevReq.Request[0] & DATA_SEND ) != 0 ) ) {
840 :                                     if( ( indadd->Word[0] & 0x0000FFFF ) == 0 ) { /* success */
841 :                                       RS_puts( "Setup Success\n" );
842 :                                       return;
843 :                                     }
844 :                                     RS_puts( "Fail in RX\r\n" );
845 :                                     return;
846 :                                   } else if ( sn_wait_data_flag ) {
847 :                                     reqtype = sn_DevReq.Request[0];
848 :                                     value =  *(unsigned short *)(& sn_DevReq.Request[2]);
849 :                                     length =  *(unsigned short *)(& sn_DevReq.Request[6]);
850 :
851 :                                     address = PCItoCPU( ( unsigned long )( indadd->Word[1] ) );
852 :                                     snow_set_descriptor( value, length, address );
853 :                                   }
854 :                           }
855 :                           /* respond */
856 :                           if(( sn_DevReq.Request[0] == 0 ) && ( sn_wait_data_flag  == 0)){
857 :                                   if(sn_set_addressflag != 0){
858 :                                           /* EndPoint Enable */
859 :                                           if( sn_TxModeFlag & ISO_TXMODE ) {
860 :                                                   outpw( USBF_EP1_CONT, ( inpw( USBF_EP1_CONT ) | ( EP1EN ) ) );
861 :                                           }
862 :                                           if( sn_TxModeFlag & BULK_TXMODE ) {
863 :                                                   outpw( USBF_EP3_CONT, ( inpw( USBF_EP3_CONT ) | ( EP3EN ) ) );
864 :                                           }
865 :                                           if( sn_TxModeFlag & INT_TXMODE ) {
866 :                                                   outpw( USBF_EP5_CONT, ( inpw( USBF_EP5_CONT ) | ( EP5EN ) ) );
867 :                                           }
868 :                                           if( sn_RxModeFlag & ISO_RXMODE ) {
869 :                                                   outpw( USBF_EP2_CONT, ( inpw( USBF_EP2_CONT ) | ( EP2EN ) ) );
870 :                                           }
871 :                                           if( sn_RxModeFlag & BULK_RXMODE ) {
872 :                                                   outpw( USBF_EP4_CONT, ( inpw( USBF_EP4_CONT ) | ( EP4EN ) ) );
873 :                                           }
874 :                                           if( sn_RxModeFlag & INT_RXMODE ) {
875 :                                                   outpw( USBF_EP6_CONT, ( inpw( USBF_EP6_CONT ) | ( EP6EN ) ) );
876 :                                           }
877 :                                   }
878 :
879 :                                   if( sn_set_addressflag != 0 ) {
880 :                                     RS_puts( "SET_ADDRESS\r\n" );
881 :                                           outpw( USBF_GMODE, ( ( inpw( USBF_GMODE ) & ( ADD_CLEAR ) ) | ( value << 16 ) ) );
882 :                                           sn_set_addressflag = 0;
883 :                                   }
884 :
885 :                                   /*********************************************
886 :                                   ** issue data transmission command.
887 :                                   *********************************************/
888 :                                   /* TX               0-Length Packet */
889 :                                   snow_tx( CTL_TXMODE, 0, ( unsigned long )sn_TxBufferDesc_Length0 );
890 :                                   RS_puts( "TX0 RX\r\n" );
891 :                           }
892 :                           return;
893 :   }
894 :
895 :   /*-----------------------------------------------------*/
896 :   /* argument:                                                                                                    */
897 :   /*                  reqtype - Device Request bmRequestType value                                    */
898 :   /*                  index           #NAME?                                                                  */
899 :   /*                  length          #NAME?                                                                  */
900 :   /* return value:    non                                                                                     */
901 :   /* use:                                                     disposal of get_status.                                                 */
902 :   /* contents:                calling by snow_rxreq().                                                */
903 :   /*-----------------------------------------------------*/
904 :   void            snow_get_status(unsigned char reqtype, unsigned short index, unsigned short length)
905 :   {
906 :                   unsigned char   stall = 0x0;
907 :                   unsigned long   reg;
908 :                   unsigned char   txData_0, txData_1;
909 :
910 :                   RS_puts( "snow_get_status\r\n" );
911 :
912 :                   reqtype = reqtype & 0x0f;
913 :                   switch( reqtype ){
914 :                           case            DEVICE:
915 :                                           /* prepare Payload Data to SDRAM */
916 :   /*                                      sn_Device_Status.Status[1] |= SELF_POWERED;*/
917 :                                           txData_0 = sn_Device_Status.Status[1];
918 :                                           txData_1 = 0x0;
919 :                                           break;
920 :
921 :                           case            INTERFACE:
922 :                                           /* prepare Payload Data to SDRAM */
923 :                                           txData_0 = 0x0;
924 :                                           txData_1 = 0x0;
925 :                                           break;
926 :
927 :                           case            ENDPOINT:
928 :                                           switch( index ){
929 :                                                   case            TXEP0:
930 :                                                                   reg = inpw( USBF_EP0_CONT );
931 :                                                                   if( reg &  EP_STALL_BIT )       stall = 1;
932 :                                                                   break;
933 :                                                   case            RXEP0:
```

```
934 :                                                                   reg = inpw( USBF_EP0_CONT );
935 :                                                                   if( reg &  EP_STALL_BIT_RX0 )                   stall = 1;
936 :                                                                   break;
937 :                                                           case    TXEP1:
938 :                                                           case    RXEP2:
939 :                                                                   break;
940 :                                                           case    TXEP3:
941 :                                                                   reg = inpw( USBF_EP3_CONT );
942 :                                                                   if( reg & EP_STALL_BIT )        stall = 1;
943 :                                                                   break;
944 :                                                           case    RXEP4:
945 :                                                                   reg = inpw( USBF_EP4_CONT );
946 :                                                                   if( reg & EP_STALL_BIT )        stall = 1;
947 :                                                                   break;
948 :                                                           case    TXEP5:
949 :                                                                   reg = inpw( USBF_EP5_CONT );
950 :                                                                   if( reg & EP_STALL_BIT )        stall = 1;
951 :                                                                   break;
952 :                                                           case    RXEP6:
953 :                                                                   reg = inpw( USBF_EP6_CONT );
954 :                                                                   if( reg & EP_STALL_BIT )        stall = 1;
955 :                                                                   break;
956 :                                                           default:
957 :                                                                   return;         /* Error */
958 :                                                   }
959 :
960 :                                                   /* prepare Payload Data to SDRAM */
961 :                                                   txData_0 = stall;
962 :                                                   txData_1 = 0x0;
963 :                                                   break;
964 :
965 :                                   default:
966 :                                                   return;
967 :                           }
968 :                           /*********************************************
969 :                           ** prepare data to SDRAM.
970 :                           *********************************************/
971 :                           if( sn_TxBufferDesc[0].Word[0] & LASTBIT ){
972 :                                   RS_puts( "TxBufferDesc[0] Full\r\n" );
973 :                                   return;
974 :                           }
975 :
976 :                           /* prepare Buffer Directory */
977 :                           sn_TxBufferDesc[0].Word[0] = ( unsigned long )( LASTBIT | DLBIT | length );
978 :                           sn_TxBufferDesc[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer );
979 :
980 :                           sn_TxBufferDesc[1].Word[0] = 0;
981 :                           sn_TxBufferDesc[1].Word[1] = 0;
982 :
983 :                           /* prepare Payload Data to SDRAM */
984 :                           sn_TxDataBuffer[0] = stall;
985 :                           sn_TxDataBuffer[1] = 0x0;
986 :
987 :                           /*********************************************
988 :                           ** issue data transmission command.
989 :                           *********************************************/
990 :                           snow_tx( CTL_TXMODE, length, ( unsigned long )sn_TxBufferDesc );
991 :                           return;
992 :           }
993 :
994 :           /*-------------------------------------------------------*/
995 :           /* argument:                                                                                                    */
996 :           /*                 reqtype - Device Request bmRequestType value                                    */
997 :           /*                 value           #NAME?                                                          */
998 :           /*                 index           #NAME?                                                          */
999 :           /* return value:  non                                                                                          */
1000 :          /* use:                                                    disposal of clear_feature.              */
1001 :          /* contents:               calling by snow_rxreq().                                                */
1002 :          /*-------------------------------------------------------*/
1003 :          void            snow_clear_feature( unsigned char reqtype, unsigned short value, unsigned short index )
1004 :          {
1005 :
1006 :                          RS_puts( "snow_clear_feature\r\n" );
1007 :
1008 :                          switch( reqtype ) {
1009 :                                  case            DEVICE:
1010 :                                                  /* DEVICE_REMOTE_WAKEUP Clear */
1011 :                                                  if( value != ( REMOTE_WAKEUP >> 1 )){
1012 :                                                          return;         /* error */
1013 :                                                  }
1014 :                                                  sn_Device_Status.Status[1] &= ~REMOTE_WAKEUP;
1015 :                                                  break;
1016 :
1017 :                                  case            INTERFACE:
1018 :                                                  /* STALL */
1019 :                                                  break;
1020 :
1021 :                                  case            ENDPOINT:
1022 :                                                  /* ENDPOINT_STALL Clear */
1023 :                                                  if( value != ( ENDPOINT_STALL >> 1 )){
1024 :                                                          return;         /* Error */
1025 :                                                  }
1026 :                                                  switch( index ){
1027 :                                                          case            TXEP0:
1028 :                                                                          outpw( USBF_EP0_CONT, ( inpw( USBF_EP0_CONT ) & ( ~EP_STALL_BIT
) ) );
1029 :                                                                          break;
1030 :                                                          case            RXEP0:
1031 :                                                                          outpw( USBF_EP0_CONT, ( inpw( USBF_EP0_CONT ) & (
~EP_STALL_BIT_RX0 ) ) );
1032 :                                                                          break;
1033 :                                                          case            TXEP1:
1034 :                                                          case            RXEP2:
1035 :                                                                          /* STALL */
1036 :                                                                          break;
```

```
1037 :                                                          case            TXEP3:
1038 :                                                                          outpw( USBF_EP3_CONT, ( inpw( USBF_EP3_CONT ) & ( ~EP_STALL_BIT
) ) );
1039 :                                                                          break;
1040 :                                                          case            RXEP4:
1041 :                                                                          outpw( USBF_EP4_CONT, ( inpw( USBF_EP4_CONT ) & ( ~EP_STALL_BIT
) ) );
1042 :                                                                          break;
1043 :                                                          case            TXEP5:
1044 :                                                                          outpw( USBF_EP5_CONT, ( inpw( USBF_EP5_CONT ) & ( ~EP_STALL_BIT
) ) );
1045 :                                                                          break;
1046 :                                                          case            RXEP6:
1047 :                                                                          outpw( USBF_EP6_CONT, ( inpw( USBF_EP6_CONT ) & ( ~EP_STALL_BIT
) ) );
1048 :                                                                          break;
1049 :                                                          default:
1050 :                                                                          return; /* Error */
1051 :                                                  }
1052 :                          }
1053 :      }
1054 :
1055 :      /*------------------------------------------------------*/
1056 :      /* argument:                                                                                                                       */
1057 :      /*              reqtype - Device Request bmRequestType value                                    */
1058 :      /*              value           #NAME?                                                                  */
1059 :      /*              index           #NAME?                                                                  */
1060 :      /* return value:  non                                                                                                      */
1061 :      /* use:                                                    disposal of set_feature.                        */
1062 :      /* contents:                       calling by snow_rxreq().                                        */
1063 :      /*------------------------------------------------------*/
1064 :      void            snow_set_feature( unsigned char reqtype, unsigned short value, unsigned short index )
1065 :      {
1066 :
1067 :                          RS_puts( "snow_set_feature\r\n" );
1068 :
1069 :                          switch( reqtype ){
1070 :                                  case            DEVICE:
1071 :                                                  /* DEVICE_REMOTE_WAKEUP set */
1072 :                                                  if( value != ( REMOTE_WAKEUP >> 1 ) ) {
1073 :                                                          return;         /* error */
1074 :                                                  }
1075 :                                                  sn_Device_Status.Status[1] |= REMOTE_WAKEUP;
1076 :                                                  break;
1077 :
1078 :                                  case            INTERFACE:
1079 :                                                  /* STALL */
1080 :                                                  break;
1081 :
1082 :                                  case            ENDPOINT:
1083 :                                                  /* ENDPOINT_STALL set */
1084 :                                                  if( value != ( ENDPOINT_STALL >> 1 ) ) {
1085 :                                                          return;         /* Error */
1086 :                                                  }
1087 :                                                  switch( index ){
1088 :                                                          case            TXEP0:
1089 :                                                                          outpw( USBF_EP0_CONT, ( inpw( USBF_EP0_CONT ) | ( EP_STALL_BIT ) )
);
1090 :                                                                          break;
1091 :                                                          case            RXEP0:
1092 :                                                                          outpw( USBF_EP0_CONT, ( inpw( USBF_EP0_CONT ) | (
EP_STALL_BIT_RX0 ) ) );
1093 :                                                                          break;
1094 :                                                          case            TXEP1:
1095 :                                                          case            RXEP2:
1096 :                                                                          /* STALL */
1097 :                                                                          break;
1098 :                                                          case            TXEP3:
1099 :                                                                          outpw( USBF_EP3_CONT, ( inpw( USBF_EP3_CONT ) | ( EP_STALL_BIT ) )
);
1100 :                                                                          break;
1101 :                                                          case            RXEP4:
1102 :                                                                          outpw( USBF_EP4_CONT, ( inpw( USBF_EP4_CONT ) | ( EP_STALL_BIT ) )
);
1103 :                                                                          break;
1104 :                                                          case            TXEP5:
1105 :                                                                          outpw( USBF_EP5_CONT, ( inpw( USBF_EP0_CONT ) | ( EP_STALL_BIT ) )
);
1106 :                                                                          break;
1107 :                                                          case            RXEP6:
1108 :                                                                          outpw( USBF_EP6_CONT, ( inpw( USBF_EP0_CONT ) | ( EP_STALL_BIT ) )
);
1109 :                                                                          break;
1110 :                                                          default:
1111 :                                                                          return; /* Error */
1112 :                                                  }
1113 :                          }
1114 :      }
1115 :
1116 :      /*
1117 :      void            snow_set_address( unsigned short address )
1118 :      {
1119 :                          outpw( USBF_GMODE, ( ( inpw( USBF_GMODE ) & ( ADD_CLEAR ) ) | ( address << 16 ) ) );
1120 :                          sn_set_addressflag = 1;
1121 :      }
1122 :      */
1123 :
1124 :      /*------------------------------------------------------*/
1125 :      /* argument:                                                                                                                       */
1126 :      /*              value           #NAME?                                                                  */
1127 :      /*              length          #NAME?                                                                  */
1128 :      /* return value:  non                                                                                                      */
1129 :      /* use:                                                    disposal of get_descriptor.                     */
1130 :      /* contents:                       calling by snow_rxreq().                                        */
1131 :      /*------------------------------------------------------*/
```

```
1132 :  void            snow_get_descriptor( unsigned short value, unsigned short length )
1133 :  {
1134 :                  int             i;
1135 :
1136 :                  RS_puts( "snow_get_descriptor\r\n" );
1137 :
1138 :                  /*********************************************
1139 :                  ** prepare data to SDRAM.
1140 :                  *********************************************/
1141 :                  if( sn_TxBufferDesc[0].Word[0] & LASTBIT ){
1142 :                          RS_puts( "TxBufferDesc Full\r\n" );
1143 :                          return;
1144 :                  }
1145 :
1146 :                  switch( value ){
1147 :                          case            DEVICE_DESC:
1148 :                                          /* prepare Payload Data to SDRAM */
1149 :                                          *(De_Desc *)sn_TxDataBuffer = sn_DeDesc;
1150 :                                          if( length > 0x12 ) {
1151 :                                                  length = 0x12;
1152 :                                          }
1153 :                                          RS_puts( "DEVICE_DESC\r\n" );
1154 :                                          break;
1155 :                          case            CONFIGURATION_DESC:
1156 :                                          /* prepare Payload Data to SDRAM */
1157 :                                          *(Co_Desc *)sn_TxDataBuffer = sn_CoDesc;
1158 :                                          if( length > 0x22 ) {
1159 :                                                  length = 0x22;
1160 :                                          }
1161 :                                          RS_puts( "CONFIGURATION_DES\r\n" );
1162 :                                          break;
1163 :                          case            STRING_DESC:
1164 :                                          /* prepare Payload Data to SDRAM */
1165 :                                          *(St_Desc *)sn_TxDataBuffer = sn_StDesc;
1166 :                                          if( length > 0x04 ) {
1167 :                                                  length = 0x04;
1168 :                                          }
1169 :                                          RS_puts( "STRING_DESC\r\n" );
1170 :                                          break;
1171 :  #if 1           // The following lines are prepared if STRING_INDEX is specified and HOST makes some request for it.
1172 :                          case            STRING_DESC_INDEX05:
1173 :                                          /* prepare Payload Data to SDRAM */
1174 :                                          *(StIn_Desc *)sn_TxDataBuffer = sn_StIndexDesc;
1175 :                                          if( length > 0x06 ) {
1176 :                                                  length = 0x12;
1177 :                                          }
1178 :                                          RS_puts( "STRING_DESC_INDEX05\r\n" );
1179 :                                          break;
1180 :                          case            STRING_DESC_INDEX02:
1181 :                                          /* prepare Payload Data to SDRAM */
1182 :                                          *(StIn_Desc *)sn_TxDataBuffer = sn_StIndexDesc;
1183 :                                          if( length > 0x06 ) {
1184 :                                                  length = 0x12;
1185 :                                          }
1186 :                                          RS_puts( "STRING_DESC_INDEX02\r\n" );
1187 :                                          break;
1188 :  #endif
1189 :                          case            INTERFACE_DESC:
1190 :                                          RS_puts( "INTERFACE_DESC\r\n" );
1191 :                                          break;
1192 :                          case            ENDPOINT_DESC:
1193 :                                          RS_puts( "ENDPOINT_DESC\r\n" );
1194 :                                          break;
1195 :  #if 1           // If HID class is used, Report information is requested after set-config.
1196 :                          case            REPORT_DESC:
1197 :                                          *(Re_Desc *)sn_TxDataBuffer = sn_ReDesc;
1198 :                                          if( length > 0x34 ) {
1199 :                                                  length = 0x34;
1200 :                                          }
1201 :                                          sn_setup_flag = 1;
1202 :                                          RS_puts( "REPORT_DESC\r\n" );
1203 :                                          break;
1204 :  #endif
1205 :                          default:
1206 :                                          RS_puts( "RETURN\r\n" );
1207 :                                          return;
1208 :                  }
1209 :                  /*********************************************
1210 :                  ** issue data transmission command.
1211 :                  *********************************************/
1212 :                  /* prepare Buffer Directory */
1213 :                  sn_TxBufferDesc[0].Word[0] = ( unsigned long )( LASTBIT | DLBIT | length );
1214 :                  sn_TxBufferDesc[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer );
1215 :
1216 :                  sn_TxBufferDesc[1].Word[0] = 0;
1217 :                  sn_TxBufferDesc[1].Word[1] = 0;
1218 :
1219 :                  /*********************************************
1220 :                  ** issue data transmission command.
1221 :                  *********************************************/
1222 :                  snow_tx( CTL_TXMODE, length, ( unsigned long )sn_TxBufferDesc );
1223 :
1224 :                  return;
1225 :  }
1226 :
1227 :  /*-------------------------------------------------------*/
1228 :  /* argument:                                                                                                    */
1229 :  /*                 value           #NAME?                                          */
1230 :  /*                 length          #NAME?                                          */
1231 :  /*                 address - Rx Buffer Descriptor Address                                          */
1232 :  /* return value:  non                                                                                          */
1233 :  /* use:                                                    disposal of set_descriptor.                     */
1234 :  /* contents:               calling by snow_rxreq().                        */
1235 :  /*-------------------------------------------------------*/
1236 :  void            snow_set_descriptor( unsigned short value, unsigned short length, unsigned long address )
```

```
1237 :  {
1238 :          int             i, j;
1239 :          unsigned short  set_size;
1240 :          unsigned char   *RxDataBuffer;
1241 :          Buffer_Desc             *RxBufferDesc;
1242 :
1243 :          RS_puts( "snow_set_descriptor\r\n" );
1244 :
1245 :          /* It reads reception data and it sets to corresponding Descriptor. */
1246 :          RxBufferDesc = (Buffer_Desc *)address;
1247 :          RxDataBuffer = ( (unsigned char *)PCItoCPU( RxBufferDesc->Word[1] ) );
1248 :          set_size = (unsigned short)( ( RxBufferDesc->Word[0] ) & 0xffff );
1249 :
1250 :          i = ( MAXP0 * ( sn_wait_data_flag - 1 ) );
1251 :
1252 :          if( ( unsigned short )( MAXP0 * sn_wait_data_flag ) >= length ){
1253 :                  sn_wait_data_flag = 0;
1254 :          }else{
1255 :                  sn_wait_data_flag++;
1256 :          }
1257 :
1258 :          switch( value ){
1259 :                  case            DEVICE_DESC:
1260 :                                  for(j = 0; j < set_size; j++){
1261 :                                          sn_DeDesc.Desc[i] = RxDataBuffer[j];
1262 :                                          i++;
1263 :                                  }
1264 :                                  break;
1265 :                  case            CONFIGURATION_DESC:
1266 :                                  for(j = 0; j < set_size; j++){
1267 :                                          sn_CoDesc.Desc[i + 60] = RxDataBuffer[j];
1268 :                                          i++;
1269 :                                  }
1270 :                                  break;
1271 :                  case            STRING_DESC:
1272 :  /*                              if( length <= 2 + N ){
1273 :                                          set_size = length;
1274 :                                  }else{
1275 :                                          set_size = 2 + N;
1276 :                                  }
1277 :                                  for( ; i < set_size; i++){
1278 :                                          sn_StDesc.Desc[i] = RxDataBuffer[i];
1279 :                                  }
1280 :  */                              break;
1281 :                  case            INTERFACE_DESC:
1282 :                                  RS_puts( "not set descriptor\r\n" );
1283 :
1284 :                                  break;
1285 :                  case            ENDPOINT_DESC:
1286 :                                  RS_puts( "not set descriptor\r\n" );
1287 :
1288 :                                  break;
1289 :                  default:
1290 :                                  return;
1291 :          }
1292 :          return;
1293 :  }
1294 :
1295 :  /*-------------------------------------------------------*/
1296 :  /* argument:                                                                                                    */
1297 :  /*                 length          #NAME?                                                  */
1298 :  /* return value:  non                                                                                           */
1299 :  /* use:                                                    disposal of get_configuration.          */
1300 :  /* contents:               calling by snow_rxreq().                                        */
1301 :  /*-------------------------------------------------------*/
1302 :  void            snow_get_configuration( unsigned short length )
1303 :  {
1304 :
1305 :          RS_puts( "snow_get_configuration\r\n" );
1306 :
1307 :          /*********************************************
1308 :          ** prepare data to SDRAM.
1309 :          *********************************************/
1310 :          if( sn_TxBufferDesc[0].Word[0] & LASTBIT ){
1311 :                  RS_puts("TxBufferDesc Full\n");
1312 :                  return;
1313 :          }
1314 :          /* prepare Payload Data to SDRAM */
1315 :          sn_TxDataBuffer[0] = sn_Value.Number[0];
1316 :
1317 :          /* prepare Buffer Directory */
1318 :          sn_TxBufferDesc[0].Word[0] = (unsigned long)( LASTBIT | DLBIT | length );
1319 :          sn_TxBufferDesc[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer );
1320 :
1321 :          sn_TxBufferDesc[1].Word[0] = 0;
1322 :          sn_TxBufferDesc[1].Word[1] = 0;
1323 :
1324 :          /*********************************************
1325 :          ** issue data transmission command.
1326 :          *********************************************/
1327 :          snow_tx( CTL_TXMODE, length, ( unsigned long )sn_TxBufferDesc );
1328 :  }
1329 :
1330 :
1331 :  /*-------------------------------------------------------*/
1332 :  /* argument:                                                                                                    */
1333 :  /*                 value - Device Request wValue value                                                  */
1334 :  /* return value:  non                                                                                           */
1335 :  /* use:                                                    disposal of set_configuration.          */
1336 :  /* contents:               calling by snow_rxreq().                                        */
1337 :  /*-------------------------------------------------------*/
1338 :  void            snow_set_configuration( unsigned char value)
1339 :  {
1340 :          int                             i = 0, j;
1341 :          unsigned char   data_size, ep;
```

```
1342 :                          unsigned long   packet_size;
1343 :
1344 :                          RS_puts( "snow_set_configuration\r\n" );
1345 :
1346 :                          sn_Value.Number[0] = ( 0xff & value );
1347 :
1348 :                          while( 1 ) {
1349 :                                  if( sn_CoDesc.Desc[ i + 1 ] == CONFIGURATION_TYPE ) {
1350 :                                          if( sn_CoDesc.Desc[ i + 5 ] == sn_Value.Number[0] ) {
1351 :                                                  /* set configu */
1352 :
1353 :                                                  /*-------------------------------------------------------*/
1354 :                                                  /* Setting the Endpoint Controll Register.                                      */
1355 :                                                  /*-------------------------------------------------------*/
1356 :                                                  for( j = 0; j < 7; j++ ) {
1357 :                                                          if( sn_CoDesc.Desc[7 * j + i + 19] == ENDPOINT_TYPE ) {
1358 :                                                                  packet_size = ( unsigned long )( ( sn_CoDesc.Desc[7 * j + i +
23]  << 8 ) | sn_CoDesc.Desc[7 * j + i + 22] );
1359 :                                                                  ep = sn_CoDesc.Desc[7 * j + i + 20];
1360 :                                                                  switch( ep ){
1361 :                                                                          case            0x81:
1362 :                                                                                  outpw( USBF_EP1_CONT, (
inpw( USBF_EP1_CONT ) | ( packet_size ) ) );
1363 :                                                                                  break;
1364 :                                                                          case            0x2:
1365 :                                                                                  outpw( USBF_EP2_CONT, (
inpw( USBF_EP2_CONT ) | ( RM2_NOR | packet_size ) ) );
1366 :                                                                                  break;
1367 :                                                                          case            0x83:
1368 :                                                                                  outpw( USBF_EP3_CONT, (
inpw( USBF_EP3_CONT ) | ( packet_size ) ) );
1369 :                                                                                  break;
1370 :                                                                          case            0x4:
1371 :                                                                                  outpw( USBF_EP4_CONT, (
inpw( USBF_EP4_CONT ) | ( RM4_ASM | packet_size ) ) );
1372 :                                                                                  break;
1373 :                                                                          case            0x85:
1374 :                                                                                  outpw( USBF_EP5_CONT, (
inpw( USBF_EP5_CONT ) | ( packet_size ) ) );
1375 :                                                                                  break;
1376 :                                                                          case            0x6:
1377 :                                                                                  outpw( USBF_EP6_CONT, (
inpw( USBF_EP6_CONT ) | ( packet_size ) ) );
1378 :                                                                                  break;
1379 :                                                                          default:
1380 :                                                                                  break;
1381 :                                                                  }
1382 :                                                          }else{
1383 :                                                                  RS_puts( "end ENDPOINT Descriptor\r\n" );
1384 :                                                                  /* error */
1385 :                                                                  break;
1386 :                                                          }
1387 :                                                  }
1388 :                                          }else{
1389 :                                                  data_size = sn_CoDesc.Desc[i + 2];              /* Total Size */
1390 :                                                  if( sn_CoDesc.Desc[i + data_size + 1] == CONFIGURATION_TYPE ){
1391 :                                                          i += data_size ;
1392 :                                                  }else{
1393 :                                                          /* error */
1394 :                                                          RS_puts(" don't set configuration\n");
1395 :                                                          break;
1396 :                                                  }
1397 :                                          }
1398 :                                  }else{
1399 :                                          /* error */
1400 :                                          RS_puts(" don't set configuration\n");
1401 :                                          break;
1402 :                                  }
1403 :                                  break;
1404 :                          }
1405 :          }
1406 :
1407 :          /*-------------------------------------------------------*/
1408 :          /* argument:                                                                    */
1409 :          /*                 length          #NAME?                                       */
1410 :          /* return value:   non                                                          */
1411 :          /* use:                                            disposal of get_interface.   */
1412 :          /* contents:               calling by snow_rxreq().                             */
1413 :          /*-------------------------------------------------------*/
1414 :          void            snow_get_interface( unsigned short length )
1415 :          {
1416 :                          RS_puts( "snow_get_interface\r\n" );
1417 :
1418 :                          /*********************************************
1419 :                          ** prepare data to SDRAM.
1420 :                          *********************************************/
1421 :                          if( sn_TxBufferDesc[0].Word[0] & LASTBIT ){
1422 :                                  RS_puts( "TxBufferDesc Full\r\n" );
1423 :                                  return;
1424 :                          }
1425 :                          /* prepare Payload Data to SDRAM */
1426 :                          sn_TxDataBuffer[0] = sn_Value.Number[1];
1427 :
1428 :                          /* prepare Buffer Directory */
1429 :                          sn_TxBufferDesc[0].Word[0] = ( unsigned long )( LASTBIT | DLBIT | length );
1430 :                          sn_TxBufferDesc[0].Word[1] = CPUtoPCI( ( unsigned long )sn_TxDataBuffer );
1431 :
1432 :                          sn_TxBufferDesc[1].Word[0] = 0;
1433 :                          sn_TxBufferDesc[1].Word[1] = 0;
1434 :
1435 :                          /*********************************************
1436 :                          ** issue data transmission command.
1437 :                          *********************************************/
1438 :                          snow_tx( CTL_TXMODE, length, ( unsigned long )sn_TxBufferDesc );
1439 :          }
```

```
1440 :
1441 :
1442 :      /*------------------------------------------------------*/
1443 :      /* argument:                                                                                                    */
1444 :      /*                value - Device Request wValue value                                                           */
1445 :      /* return value:  non                                                                                           */
1446 :      /* use:                                                    disposal of set_interface.                           */
1447 :      /* contents:                 calling by snow_rxreq().                                                           */
1448 :      /*------------------------------------------------------*/
1449 :      void              snow_set_interface( unsigned char value )
1450 :      {
1451 :                        int               i = 0, j;
1452 :                        unsigned char     data_size, ep;
1453 :                        unsigned long     packet_size;
1454 :
1455 :                        RS_puts( "snow_set_interface\r\n" );
1456 :
1457 :                        sn_Value.Number[1] = ( value & 0xff );
1458 :
1459 :                        while(1){
1460 :                                  if( sn_CoDesc.Desc[i + 5] == sn_Value.Number[0] ) {
1461 :                                            while( 1 ) {
1462 :                                                                      if(( sn_CoDesc.Desc[i + 10] == INTERFACE_TYPE ) && ( sn_CoDesc.Desc[i + 12] ==
sn_Value.Number[1] ) ) {
1463 :                                                                                /* select an alternate setting for the specified interface */
1464 :
1465 :                                                                                /*------------------------------------------------------*/
1466 :                                                                                /* Setting the Endpoint Controll Register.              */
1467 :                                                                                /*------------------------------------------------------*/
1468 :                                                                                for( j = 0; j < 7; j++ ){
1469 :                                                                                          if( sn_CoDesc.Desc[7 * j + i + 19] == ENDPOINT_TYPE ){
1470 :                                                                                                    packet_size = ( unsigned long )( (
sn_CoDesc.Desc[7 * j + i + 23]  << 8 ) | sn_CoDesc.Desc[7 * j + i + 22] );
1471 :                                                                                                    ep = sn_CoDesc.Desc[7 * j + i + 20];
1472 :                                                                                                    switch( ep ){
1473 :                                                                                                              case      0x81:
1474 :                                                                                                                        outpw(
USBF_EP1_CONT, ( inpw( USBF_EP1_CONT ) | ( packet_size ) ) );
1475 :                                                                                                                        break;
1476 :                                                                                                              case      0x2:
1477 :                                                                                                                        outpw(
USBF_EP2_CONT, ( inpw( USBF_EP2_CONT ) | ( RM2_NOR | packet_size ) ) );
1478 :                                                                                                                        break;
1479 :                                                                                                              case      0x83:
1480 :                                                                                                                        outpw(
USBF_EP3_CONT, ( inpw( USBF_EP3_CONT ) | ( packet_size ) ) );
1481 :                                                                                                                        break;
1482 :                                                                                                              case      0x4:
1483 :                                                                                                                        outpw(
USBF_EP4_CONT, ( inpw( USBF_EP4_CONT ) | ( RM4_ASM | packet_size ) ) );
1484 :                                                                                                                        break;
1485 :                                                                                                              case      0x85:
1486 :                                                                                                                        outpw(
USBF_EP5_CONT, ( inpw( USBF_EP5_CONT ) | ( packet_size ) ) );
1487 :                                                                                                                        break;
1488 :                                                                                                              case      0x6:
1489 :                                                                                                                        outpw(
USBF_EP6_CONT, ( inpw( USBF_EP1_CONT ) | ( packet_size ) ) );
1490 :                                                                                                                        break;
1491 :                                                                                                              default:
1492 :                                                                                                                        break;
1493 :                                                                                                    }
1494 :                                                                                          }else{
1495 :                                                                                                    RS_puts( "not ENDPOINT Descriptor\r\n" );
1496 :                                                                                                    /* error */
1497 :                                                                                                    break;
1498 :                                                                                          }
1499 :                                                                                }
1500 :                                                                                break;
1501 :                                                                      }else if( sn_CoDesc.Desc[i + 10] == INTERFACE_TYPE ) {
1502 :                                                                                data_size = ( sn_CoDesc.Desc[i + 13] * 7 ) + 9;
1503 :                                                                                /* INT_Desc Size + ( EP_Desc Size * EP_Desc Num ) - 9 */
1504 :                                                                                i += data_size;
1505 :                                                                      }else{
1506 :                                                                                RS_puts( "not INTERFACE Descriptor\r\n" );
1507 :                                                                                break;
1508 :                                                                      }
1509 :                                            }
1510 :                                            break;
1511 :                                  }else{
1512 :                                            data_size = sn_CoDesc.Desc[i + 2];          /* Total Size */
1513 :                                            if( sn_CoDesc.Desc[data_size + 1] == CONFIGURATION_TYPE ){
1514 :                                                      i += ( data_size - 1 );
1515 :                                            }else{
1516 :                                                      RS_puts( "not CONFIGURATION Descriptor\r\n" );
1517 :                                                      break;
1518 :                                            }
1519 :                                  }
1520 :                        }
1521 :      }
1522 :
1523 :
1524 :      /*------------------------------------------------------*/
1525 :      /* argument:                 non                                                                                */
1526 :      /* return value:  non                                                                                           */
1527 :      /* use:                                                    disposal of synch_frame.                             */
1528 :      /* contents:                 calling by snow_rxreq().                                                           */
1529 :      /*------------------------------------------------------*/
1530 :      void              snow_synch_frame( void )
1531 :      {
1532 :                        return;
1533 :      }
1534 :
1535 :      /*----------------------------------------------------------------------*/
1536 :      /*----------------------------------------------------------------------*/
```

```
1537 :  /* Rx Section                                                                                                  */
1538 :  /*-------------------------------------------------------------------------*/
1539 :  /*-------------------------------------------------------------------------*/
1540 :
1541 :  /*-------------------------------------------------------------------------*/
1542 :  /* argument:                  non
1543 :  /* return value:  non
1544 :  /* use:                                                    Rx end processing
1545 :  /* contents:                  Checking that there is any error or not.
1546 :  /*                                                                  Checking received EP.
1547 :  /*                                                                  Giving the data to bridge.
1548 :  /*-------------------------------------------------------------------------*/
1549 :  void            snow_rxdone( void )
1550 :  {
1551 :                  Rx_Indication * p_MailBox;
1552 :                  unsigned long word0_indication, word1_address;
1553 :
1554 :                  Buffer_Desc * p_desc;
1555 :                  unsigned long   pnum;
1556 :
1557 :                  RS_puts( "snow_rxdone\r\n" );
1558 :
1559 :                  /* Getting the latest RxIndication */
1560 :                  p_MailBox = (Rx_Indication *)PCItoCPU( inpw( USBF_RxMail_RD_ADDR ) );
1561 :                  /* Storing the Indication data into variable for calculation. */
1562 :                  word0_indication = p_MailBox->Word[0];
1563 :                  word1_address    = PCItoCPU( p_MailBox->Word[1] );
1564 :
1565 :                  p_desc = (Buffer_Desc *)word1_address;
1566 :                  pnum = ( word0_indication & 0xe0000000 );
1567 :                  switch( pnum ) {
1568 :                          case EPN0_IN_RXINDI:
1569 :                          case EPN6_IN_RXINDI:
1570 :                                  p_desc->Word[0] = ( p_desc->Word[0] | ( POOLNUM_0 ) );
1571 :                                  break;
1572 :                          case EPN2_IN_RXINDI:
1573 :                                  p_desc->Word[0] = ( p_desc->Word[0] | ( POOLNUM_1 ) );
1574 :                                  break;
1575 :                          case EPN4_IN_RXINDI:
1576 :                                  p_desc->Word[0] = ( p_desc->Word[0] | ( POOLNUM_2 ) );
1577 :                                  break;
1578 :                          default:
1579 :                                  /* error */
1580 :                                  RS_puts( "Pool doesn't agree\r\n" );
1581 :                                  break;
1582 :                  }
1583 :
1584 :                  /*-------------------------------------------------------------------------*/
1585 :                  /* If it's IBUS error, do nothing .                                                           */
1586 :                  /*-------------------------------------------------------------------------*/
1587 :                  if( word0_indication & IBUS_ERROR_IN_RXINDI )
1588 :                  {
1589 :                          RS_puts(" IBUS_ERROR_IN_RXINDI\n ");
1590 :                  }
1591 :
1592 :                  /*-------------------------------------------------------------------------*/
1593 :                  /* If it's error about EP2, return the DataBuffer.                                            */
1594 :                  /*-------------------------------------------------------------------------*/
1595 :                  else if( word0_indication & ( DATA_CORRUPTION_EP2 | BUFFER_OVERRUN_EP2
1596 :                                                                                                  | CRC_ERROR_EP2 |
BIT_STUFFING_ERROR_EP2 ) )
1597 :                  {
1598 :                          if( word0_indication & DATA_CORRUPTION_EP2 )
1599 :                                  RS_puts("DATA_CORRUPTION_EP2\n");
1600 :                          if( word0_indication & BUFFER_OVERRUN_EP2 )
1601 :                                  RS_puts("BUFFER_OVERRUN_EP2\n");
1602 :                          if( word0_indication & CRC_ERROR_EP2 )
1603 :                                  RS_puts("CRC_ERROR_EP2\n");
1604 :                          if( word0_indication & BIT_STUFFING_ERROR_EP2 )
1605 :                                  RS_puts("BIT_STUFFING_ERROR_EP2\n");
1606 :
1607 :                          snow_add_buffer( (Buffer_Desc *)word1_address );
1608 :                  }
1609 :
1610 :                  /*-------------------------------------------------------------------------*/
1611 :                  /* If it's reception at EP0, go devicerequest operation.                                      */
1612 :                  /*-------------------------------------------------------------------------*/
1613 :                  else if( ( word0_indication & EPN_MASK ) == EPN0_IN_RXINDI ) {
1614 :                          snow_rxreq( p_MailBox );
1615 :                          if( word0_indication & 0x0000ffff ) {
1616 :                                  snow_add_buffer( (Buffer_Desc *)word1_address );
1617 :                          }
1618 :                  }
1619 :
1620 :                  /*-------------------------------------------------------*/
1621 :                  /* Give Data to Bridge.                                                                       */
1622 :                  /*-------------------------------------------------------*/
1623 :                  else {
1624 :
1625 :                  }
1626 :
1627 :                  /*-------------------------------------------------------------------------*/
1628 :                  /* Wrighting the address of next Mailbox into RMRA Register.                  */
1629 :                  /*-------------------------------------------------------------------------*/
1630 :                  if( p_MailBox+1 == &sn_RxMailBox[ SN_RX_MLBNUM ] ) {
1631 :                          outpw( USBF_RxMail_RD_ADDR, CPUtoPCI( ( ( unsigned long )&sn_RxMailBox[0] ) ) );
1632 :                  } else {
1633 :                          outpw( USBF_RxMail_RD_ADDR, (Rx_Indication *)CPUtoPCI( ( unsigned long )( p_MailBox+1 ) ) );
1634 :                  }
1635 :  }
1636 :
1637 :
1638 :  /*-------------------------------------------------------------------------*/
1639 :  /*-------------------------------------------------------------------------*/
1640 :  /* Tx Section                                                                                                 */
```

```
1641 :  /*----------------------------------------------------------------------*/
1642 :  /*----------------------------------------------------------------------*/
1643 :
1644 :  /*----------------------------------------------------------------------*/
1645 :  /* argument:                                                                                    */
1646 :  /*                  mode            - Tx mode                                                   */
1647 :  /*                  DataSize - data size
1648 :  /*                  Address         - Buffer Descriptor address                                 */
1649 :  /* return value:  The result of doing transmission command issue.                   */
1650 :  /* use:                                                    issue data transmission command.    */
1651 :  /* contents:                  called by data transmission command issue request.   */
1652 :  /*                                                                 U_TEPSR and U_CMR Register Busy check.  */
1653 :  /*                                                                 issue data transmission command.        */
1654 :  /*----------------------------------------------------------------------*/
1655 :  int                        snow_tx( int mode, unsigned long size, unsigned long BufferDesc )
1656 :  {
1657 :                  int             intLevel, szlp = 0;
1658 :                  unsigned long   command, reg, packet_size;
1659 :
1660 :                  RS_puts( "snow_tx\r\n" );
1661 :
1662 :                  /* endpoint value =  EP0 -> 0, EP1-> 1, EP3-> 2, EP5-> 4 */
1663 :                  switch( mode ) {
1664 :                          case            CTL_TXMODE:
1665 :                                          reg = inpw( USBF_EP0_CONT );
1666 :                                          szlp = 1;
1667 :                                          command = CMD_SEND_EP0;
1668 :                                          break;
1669 :                          case            ISO_TXMODE:
1670 :                                          reg = inpw( USBF_EP1_CONT );
1671 :                                          if( reg & 0x00080000 ) szlp = 1;
1672 :                                          packet_size = (reg & 0x000007ff);
1673 :                                          command = CMD_SEND_EP1;
1674 :                                          break;
1675 :                          case            BULK_TXMODE:
1676 :                                          reg = inpw( USBF_EP3_CONT );
1677 :                                          packet_size = (reg & 0x000000ff);
1678 :                                          if(reg & 0x00080000) szlp = 1;
1679 :                                          command = CMD_SEND_EP3;
1680 :                                          break;
1681 :                          case            INT_TXMODE:
1682 :                                          reg = inpw( USBF_EP5_CONT );
1683 :                                          packet_size = (reg & 0x000000ff);
1684 :                                          command = CMD_SEND_EP5;
1685 :                                          break;
1686 :                          default:
1687 :                                          return          SN_TXMODE_ERROR;        /* Error */
1688 :                  }
1689 :                  if( ( reg >> 31 ) == 0 ) {
1690 :                                  /*              EndPoint desable :not send Data             */
1691 :                                  return          SN_TXEPN_DESABLE;       /* Error */
1692 :                  }
1693 :                  /* U_TEPSR BUSY check */
1694 :                  if( ( inpw( USBF_TxEP_STAT ) & ( TX_FULL_BIT << ( mode * 8 ) ) ) == 0 ){
1695 :
1696 :                          /* not BUSY : The squeeze prohibition */
1697 :  //                      intLevel = intLock();
1698 :
1699 :                          while( 1 ) {
1700 :                                  if( ( inpw( USBF_CMD ) & CMR_BUSY ) == 0 ) {
1701 :                                          break;
1702 :                                  }
1703 :                          }
1704 :
1705 :                          outpw( USBF_CMD_ADDRESS, CPUtoPCI( BufferDesc ) );
1706 :                          outpw( USBF_CMD, ( command | size ) );
1707 :
1708 :                          if( szlp == 0 ) {
1709 :                                  if( ( size != 0 ) && ( ( size % packet_size ) == 0 ) ) {
1710 :                                  outpw( USBF_CMD_ADDRESS, CPUtoPCI( ( unsigned long )sn_TxBufferDesc_Length0 ) );
1711 :                                  outpw( USBF_CMD, command );
1712 :                                  }
1713 :                          }
1714 :
1715 :                          /* The squeeze permission */
1716 :  //                      intUnlock( intLevel );
1717 :
1718 :                  } else {
1719 :                          return SN_TEPSR_BUSY;   /* Error */
1720 :                  }
1721 :                  return          SN_TX_SUCCESS;
1722 :  }
1723 :
1724 :  /*----------------------------------------------------------------------*/
1725 :  /* argument:                  non                                                              */
1726 :  /* return value:  non                                                                          */
1727 :  /* use:                                    The transmission ending processing.                 */
1728 :  /* contents:      The transmission check.                                                      */
1729 :  /*                                                 calling function for open of Buffer Descriptor.         */
1730 :  /*----------------------------------------------------------------------*/
1731 :  void            snow_txdone( void )
1732 :  {
1733 :                  Tx_Indication   *TxMailBox;
1734 :
1735 :                  RS_puts( "snow_txdone\r\n" );
1736 :
1737 :                  /*********************************************
1738 :                  ** The transmission check
1739 :                  *********************************************/
1740 :                  /* The content confirmation of TxMailBox Indication */
1741 :                  TxMailBox = (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_RD_ADDR ) );
1742 :
1743 :                  if( TxMailBox->Word[0] & IBUS_ERROR ) {
1744 :                          RS_puts( "IBusError\r\n" );
1745 :                  } if( TxMailBox->Word[0] & EP1_BUFF_UNDERRUN ) {
```

```
1746 :                                  RS_puts( "EP1BuffUnderrun\r\n" );
1747 :                  }
1748 :
1749 :                  /*********************************************
1750 :                  ** update the Mail Box Read Address
1751 :                  *********************************************/
1752 :                  if( ( TxMailBox + 1 ) >= ( (Tx_Indication *)PCItoCPU( inpw( USBF_TxMail_BTM_ADDR ) ) ) ) {
1753 :                                  outpw( USBF_TxMail_RD_ADDR, inpw( USBF_TxMail_START_ADDR ) );
1754 :                  } else {
1755 :                                  outpw( USBF_TxMail_RD_ADDR, CPUtoPCI( (unsigned long)( TxMailBox + 1 ) ) );
1756 :                  }
1757 :
1758 :
1759 :                  /* It returns processing to the bridge processing part */
1760 :                  if( TxMailBox->Word[0] & TX_EPN ){
1761 :      //                          snow_txdone_retbridge();
1762 :                  } else {
1763 :                                  /* Clear Buffer Directory */
1764 :                                  sn_TxBufferDesc[0].Word[0] = 0x0;
1765 :                  }
1766 :
1767 :                  return;
1768 :      }
```

# APPENDIX B. REVISION HISTORY

| Description | Document Number | Issued on |
|---|---|---|
| Newly created | SUD-DT-02-0067-E | Dec. 9, 2002 |
| • Addition of description related to VR4181A USBFU software reset in (8) under **6.1.4 Operation after USB cable connection**<br>• Addition of **APPENDIX A SOURCE CODE OF SAMPLE PROGRAM** | SUD-DT-03-0109-E | Apr. 14, 2003 |