To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010 Renesas Electronics Corporation

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

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

Notice

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

- "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
- "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
- "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
- 8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
- 12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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



Application Note

78K0R/KC3-L, 78K0R/KE3-L (On-Chip USB Controller)

16-bit Single-Chip Microcontroller USB CDC (Communication Device Class) Driver

μPD78F1022 μPD78F1023 μPD78F1024 μPD78F1025 μPD78F1026

© Renesas Electronics Corporation 2010

Document No R01AN0003EJ0100(U20312EJ1V0AN00) Date Published 2010/4/26 [Memo]

MINICUBE is a registered trademark of NEC Electronics Corporation

Windows, Windows XP are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

PC/AT is a trademark of International Business Machines Corporation.

NOTES FOR CMOS DEVICES

- (1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).
- (2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.
- (3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.
- (4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.
- (5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.
- (6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

PREFACE

Readers

This application note is intended for users, who understand the features of the 78K0R/KC3-L, KE3-L, and who try to design and develop the application system and application program using this product.

| Target | products | are | aiven | below |
|--------|----------|-----|-------|-------|
| rarget | producia | are | given | DCIOW |

| Generic Name | Standard product | USB controller built-in product |
|-----------------|--|---------------------------------|
| 78K0R/KC3-L | μPD78F1000, 78F1001, 78F1002, 78F1003 | µPD78F1022, 78F1023, 78F1024 |
| 78K0R/KE3-L | µPD78F1007, 78F1008, 78F1009 | µPD78F1025, 78F1026 |

| Purpose | This manual is intended to give users an understanding of the functions mentioned in following organization. | | |
|-------------------------|---|--|--|
| Organization | This application note i | is broadly divided into the following sections. | |
| | o An overview of theo The specifications | for the sample driver for the sample application ronment | |
| How to Read This Manual | | readers of this application note have general of electrical engineering, logic circuits, and | |
| | To learn about the hardware features and electrical specifications of the 78K0R/KC3-L, KE3-L. → See the separately provided 78K0R/KC3-L, KE3-L Hardware User's Manual. | | |
| | | structions of the 78K0R/KC3-L, KE3-L parately provided 78K0R Architecture User's | |
| Conventions | Data significance: | Higher digits on the left and lower digits on | |
| | the right Note : | Footnote for item marked with Note in the text | |
| | | Information requiring particular attention Supplementary information on: Binary or decimal XXXX Hexadecimal 0xXXXX er of 2 (address space, memory capacity) K (kilo): 2 ¹⁰ = 1,024 | |
| | | M (mega): $2^{20} = 1,024^2$ | |
| | | G (giga): $2^{30} = 1,024^3$ | |
| | | T (tera): 2 ⁴⁰ = 1,024 ⁴ | |
| | | P (peta): $2^{50} = 1,024^{5}$ | |
| | | $E (exa): 2^{60} = 1,024^{6}$ | |

CONTENTS

| CHAP | TER 1 OVERVIEW | 7 |
|---------|---|----|
| 1. 1 | Overview | 7 |
| 1. 1. 1 | Features of the USB function controller | 7 |
| 1. 1. 2 | Features of the sample driver | 8 |
| 1. 1. 3 | Files included in the sample driver | |
| 1. 2 | Overview of 78K0R/Kx3-L | 9 |
| 1. 2. 1 | Applicable products | 9 |
| 1. 2. 2 | Features | 10 |

CHAPTER 2 OVERVIEW OF USB...... 11

| 2. 1 | Transfer Format | 11 |
|---------|-----------------|----|
| 2. 2 | Endpoints | 12 |
| 2. 3 | Device Class | 12 |
| 2.4 | Requests | 13 |
| 2. 4. 1 | Турез | 13 |
| 2.4.2 | Format | 14 |
| 2. 5 | Descriptor | 15 |
| 2. 5. 1 | Турез | 15 |
| 2. 5. 2 | Format | 16 |

| 3. 1 | Overview | 18 |
|---------|---|----|
| 3. 1. 1 | Features | |
| 3. 1. 2 | Supported requests | |
| 3. 1. 3 | Descriptor settings | 20 |
| 3. 2 | Operation of Each Section | 23 |
| 3. 2. 1 | CPU Initialization | 25 |
| 3. 2. 2 | USB function controller initialization processing | |
| 3. 2. 3 | INTUSB interrupt process | |
| 3. 3 | Function Specifications | 31 |
| 3. 3. 1 | Functions | 31 |
| 3. 3. 2 | Correlation of the functions | 32 |
| 3. 3. 3 | Function features | |

CHAPTER 4 SAMPLE APPLICATION SPECIFICATIONS 62

| 4. 1 | Overview | 62 |
|------|-----------------|----|
| 4. 2 | Operation | 62 |
| 4.3 | Using Functions | 63 |

CHAPTER 5 DEVELOPMENT ENVIRONMENT 65

| 5. 1 | Development environment | 65 |
|---------|---------------------------------------|----|
| 5. 1. 1 | Program development | |
| 5. 1. 2 | Debugging | |
| 5. 2 | Setting up the Environment | |
| 5. 2. 1 | Preparing the host environment | |
| 5. 2. 2 | Setting up the target environment | 74 |
| 5.3 | On-Chip Debugging | |
| 5. 3. 1 | Generating a load module | |
| 5. 3. 2 | Loading and executing the load module | |
| 5.4 | Checking the Operation | |

CHAPTER 6 USING THE SAMPLE DRIVER 86

| 6. 1 | Overview | 86 |
|---------|---|----|
| 6. 2 | Customizing the Sample Driver | 87 |
| 6. 2. 1 | Application section | 87 |
| 6. 2. 2 | Setting up the registers | 88 |
| 6. 2. 3 | Descriptor information | 88 |
| 6. 2. 4 | Setting up the virtual COM port host driver | 89 |
| 6. 3 | Using Functions | 92 |

| 7.1 | Overview | . 94 |
|---------|----------------|------|
| | | |
| 7. 1. 1 | Features | 94 |
| 7.2 | Specifications | 95 |
| | | |

CHAPTER 1 OVERVIEW

This application note describes the USB (communication device class) sample driver created for the USB function controller incorporated in the 78K0R/KC3-L, 78K0R/KE3-L (78K0R/Kx3-L) microcontrollers. This application note provides the following information:

- The specifications for the sample driver
- Information about the environment used to develop an application program by using the sample driver
- The reference information provided for using the sample driver

This chapter provides an overview of the sample driver and describes the microcontrollers for which the sample driver can be used.

1.1 Overview

1.1.1 Features of the USB function controller

The USB function controller that is incorporated in the 78K0R/Kx3-L and is to be controlled by the sample driver has the following features:

- Conforms to the Universal Serial Bus Rev. 2.0 Specification
- Operates as a full-speed (12 Mbps) device.
- Includes the following endpoints:

Table 1-1 Configuration of the Endpoints of the 78K0R/Kx3-L

| Endpoint Name | FIFO Size (Bytes) | Transfer Type | Remark |
|-----------------|-------------------|---------------------------|---------------|
| Endpoint0 Read | 64 | Control transfer (IN) | Single buffer |
| | | | configuration |
| Endpoint0 Write | 64 | | Single buffer |
| | | Control transfer (OUT) | configuration |
| Endpoint1 | 64x2 | | Dual-buffer |
| | | Bulk transfer 1 (IN) | configuration |
| Endpoint2 | 64x2 | | Dual-buffer |
| | | Bulk transfer 1 (OUT) | configuration |
| Endpoint3 | 64x2 | Bulk transfer 2 (IN) | Dual-buffer |
| | | | configuration |
| Endpoint4 | 64x2 | | Dual-buffer |
| | | Bulk transfer 2 (OUT) | configuration |
| Endpoint7 | 64 | Interrupt transfer 1 (IN) | Single buffer |
| | | | configuration |
| Endpoint8 | 64 | Interrupt transfer 2 (IN) | Single buffer |
| | | | configuration |

- Automatically responds to standard USB requests (except some requests).
- Can operate as a bus-powered device or self-powered device^{Note 1}
- The internal or external clock can be selected Note 2
 - Internal clock: 20 MHz External clock divided by 5 internal clock multiplied by 12 internal clock / 16 MHz external clock divided by 4 internal clock multiplied by 12 internal clock.
 - 12 MHz external clock divided by 2 internal multiplied by 8 internal (48 MHz)
 - **Notes 1.** The sample driver selects bus power.
 - 2. The sample driver selects the internal clock.

1.1.2 Features of the sample driver

The USB communication device class sample driver for the 78K0R/Kx3-L has the features below. For details about the features and operations, see **CHAPTER 3 SAMPLE DRIVER SPECIFICATIONS**.

- Conforms to the USB communication device class Ver.1.1 Abstract Control Model
- Operates as a virtual COM device
- Exclusively uses the following amounts of memory (excluding the vector table):
 - ROM:About 3.0 KB
 - RAM:About 0.4 KB

1. 1. 3 Files included in the sample driver

The sample driver includes the following files:

| Table 1-2 Files Included in the Sample Drive | ər |
|--|----|
|--|----|

| fer, control transfer |
|-----------------------|
| fer, control transfer |
| |
| ing |
| |
| |
| e declarations |
| |
| |
| |
| |
| |

Remarks In addition, the project-related files generated when creating a development environment by using the PM+ (an integrated development tool made by NEC Electronics) are also included. For details see 5.2.1 **Preparing the host environment.**

1.2 Overview of 78K0R/Kx3-L

This section describes the 78K0R/KC3-L, KE3-L which are controlled by using the sample driver. The 78K0R/KC3-L, KE3-L are products in the low-power series of single chip 78K0R microcontroller, made by NEC Electronics. They use 78K0R CPU core and have peripheral functions such as ROM/RAM, timers/counters, POC/LVI, a serial interface, A/D converter, DMA controller, USB function controller. For details, see the 78K0R/KC3-L, KE3-L **USB controller built-in products Hardware User's manual.**

1.2.1 Applicable products

The sample driver can be used for the following products.

Table 1-2 78K0R/Kx3-L Products

| Generic Name | Part Number | Internal Memory | | Incorporated USB Function | Inte | rrupt |
|--------------|-------------|-----------------|------|---------------------------|----------|----------|
| | | Flash | RAM | | Internal | External |
| | | Memory | | | | |
| 78K0R/KC3-L | μ PD78F1022 | 64 KB | 6 KB | Function controller | 36 | 7 |
| (48pin) | μ PD78F1023 | 96KB | 8 KB | Function controller | 36 | 7 |
| | μ PD78F1024 | 128KB | 8 KB | Function controller | 36 | 7 |
| 78K0R/KE3-L | μ PD78F1025 | 96KB | 8 KB | Function controller | 41 | 11 |
| (64pin) | μ PD78F1026 | 128KB | 8 KB | Function controller | 41 | 11 |

Caution: In this application note, all target microcontrollers are collectively indicated as the 78K0R/Kx3-L, unless distinguishing between them is necessary.

1.2.2 Features

The main features of 78K0R/Kx3-L are as follows. For details, see 78K0R/Kx3-L user's manual.

Memory space:

• 1M byte linear address space (for programs and data)

Internal memory

- RAM:6K/ 8K byte
- Flash memory : 64K/ 96K/ 128K byte

Multiplication/division function

- 16 bit x16 bit = 32 bit(multiplication)
- 32 bit ÷ 32 bit = 32 bit (division)

Key interrupt

- 4 channels
- 8 channels

DMA controller

• 2 channels

Serial interface

- CSI:1 channel/ UART :1 channel
- CSI:1 channel/UART:1 channel/simple I2C: 1channel
- CSI:1 channel note/UART:1 channel note/simple I2C: 1channel note
- UART(for LIN-bus):1 channel
- I2C:1 channel

USB controller

• USB function (full speed):1 channel

A/D converter

• 10 bit resolution A/D converter(AVREF = 1.8~3.6 V):8 channel

Power supply voltage

- VDD = 1.8~3.6 V(when USB is not used)
- VDD = $3.0 \sim 3.6$ V(when USB is used)

Clock output/buzzer output

- 2.44 kHz, 4.88 kHz, 9.76 kHz, 1.25 MHz, 2.5 MHz, 5 MHz, 10 MHz(peripheral hardware clock:at f_{MAIN} = 20 MHz operation)
- 256 Hz, 512 Hz, 1.024 kHz, 2.048 kHz, 4.096 kHz, 8.192 kHz, 16.384 kHz, 32.768 kHz (Subsystem clock: at f_{SUB} = 32.768 kHz operation)

With built-in on chip debugging function

Note: only 78K0R/KE3-L

CHAPTER 2 OVERVIEW OF USB

This chapter provides an overview of the USB standard, which the sample driver conforms to.

USB (Universal Serial Bus) is an interface standard for connecting various peripherals to a host computer by using the same type of connector. The USB interface is more flexible and easier to use than older interfaces in that it can connect up to 127 devices by adding a branching point known as a hub and supports the hot-plug feature, which enables devices to be recognized by Plug & Play. The USB interface is provided in most current computers and has become the standard for connecting peripherals to a computer.

The USB standard is formulated and managed by the USB Implementers Forum (USB-IF). For details about the USB standard, see the official USB-IF website (<u>www.usb.org</u>).

2.1 Transfer Format

Four types of transfer formats (control, bulk, interrupt and isochronous) are defined in the USB standard. Table 2-1 shows the features of each transfer format.

| Item | Transfer Format | Control Transfer | Bulk Transfer | Interrupt Transfer | Isochronous |
|----------------------------|------------------------|--|---|--|---|
| | | | | | Transfer |
| Feature | | Transfer format used to exchange information required for controlling peripheral devices | Transfer format used to aperiodically handle large amounts of data | Periodic data transfer format that has a low band width | Transfer format used for a real-time transfer |
| Specifiable packet size | High speed 480 Mbps | 64 bytes | 512 bytes | 1 to 1,024 bytes | 1 to 1,024 bytes |
| | Full speed 12 Mbps | 8, 16, 32, or 64 bytes | 8, 16, 32, or 64 bytes | 1 to 64 bytes | 1 to 1,023 bytes |
| | Low speed 1.5 Mbps | 8 bytes | _ | 1 to 8 bytes | _ |
| Transfer priority | | 3 | 3 | 2 | 1 |

Table 2-1 USB Transfer Format

2.2 Endpoints

An endpoint is an information unit that is used by the host device to specify a communicating device and is specified using a number from 0 to 15 and a direction (IN or OUT). An endpoint must be provided for every data communication path that is used for a peripheral device and cannot be shared by multiple communication paths^{Note}. For example, a device that can write to and read from an SD card and print out documents must have a separate endpoint for each purpose. Endpoint 0 is used to control transfers for any type of device.

During data communication, the host uses a USB device address, which specifies the device, and an endpoint (a number and direction) to specify the communication destination in the device.

Peripheral devices have buffer memory that is a physical circuit to be used for the endpoint and functions as a FIFO that absorbs the difference in speed of the USB and communication destination (such as memory).

Note An endpoint can be exclusively switched by using the alternative setting.

2.3 Device Class

Various device classes, such as the mass storage class (MSC), communication device class (CDC), and human interface device class (HID) are defined according to the functions of the peripheral devices connected via USB (the function devices). A common host driver can be used if the connected devices conform to the standard specifications of the relevant device class, which is defined by a protocol. The Communication Device Class (CDC) is intended for communication devices connected to hosts, such as modems, FAX machines and network cards. The class is increasingly used for devices that are used for USB-to-serial conversion performing UART communication with a computer, because recent computers do not have an RS-232C interface. Note that a different CDC model is defined depending on the device to connect. The sample driver uses the Abstract Control Model.

2.4 Requests

For the USB standard, communication starts with the host issuing a command, known as a request, to a function device. A request includes data such as the direction and type of processing and address of the function device.

2.4.1 Types

There are three types of requests: standard requests, class requests and vendor requests. The sample driver supports the following requests.

Standard requests

Standard requests are used for all USB-compatible devices.

| Request Name | Target Descriptor | Overview |
|-------------------|-------------------|---|
| GET_STATUS | Device | Reads the settings of the power supply (self or bus) and |
| | | remote wakeup. |
| | Endpoint | Reads the halt status. |
| CLEAR_FEATURE | Device | Clears remote wakeup. |
| | Endpoint | Cancels the halt status (DATA PID = 0). |
| SET_FEATURE | Device | Specifies remote wakeup or test mode. |
| | Endpoint | Specifies the halt status. |
| GET_DESCRIPTOR | Device | Reads the target descriptor. |
| | Configuration | |
| | string | |
| SET_DESCRIPTOR | Device | Changes the target descriptor (optional). |
| | Configuration | |
| | string | |
| GET_CONFIGURATION | Device | Reads the currently specified configuration values |
| SET_CONFIGURATION | Device | Specifies the configuration values. |
| GET_INTERFACE | Interface | Reads the alternatively specified value among the currently |
| | | specified values of the target interface. |
| SET_INTERFACE | Interface | Specifies the alternatively specified value of the target |
| | | interface. |
| SET_ADDRESS | Device | Specifies the USB address |
| SYNCH_FRAME | Endpoint | Reads frame-synchronous data. |

Table 2-2 Standard Requests

Class requests

Class requests are unique to device classes. For the sample driver, processing to respond to class requests that support the CDC Abstract Control Model is implemented. The following requests can be responded to

- SendEncapsulatedCommand This request is used to issue commands in the format of the protocol for controlling the communication class interface.
- GetEncapsulatedResponse This request is used to request a response in the format of the protocol for controlling the communication class interface.
- SetLineCoding This request is used to specify the serial communication format.
- GetLineCoding This request is used to acquire the communication format settings on the device side.
- SetControlLineState This request is used for RS-232/V.24 format control signals.

2.4.2 Format

USB requests have an 8-byte length and consist of the following fields.

| Offset | Field | | Description |
|--------|---------------|--------------|---|
| 0 | bmRequestType | | Request attribute |
| | | Bit 7 | Data transfer direction |
| | | Bits 6 and 5 | Request type |
| | | Bits 4 to 0 | Target descriptor |
| 1 | bRequest | | Request code |
| 2 | wValue | Lower | Any value used by the request |
| 3 | | Higher | |
| 4 | wIndex | Lower | Index or offset used by the request |
| 5 | | Higher | |
| 6 | wLength | Lower | Number of bytes transferred at the data |
| | | | stage |
| 7 | | Higher | (the data length) |

Table 2-3 USB Request Format

2.5 Descriptor

For the USB standard, a descriptor is information that is specific to a function device and is encoded in a specified format. A function device transmits a descriptor in response to a request transmitted from the host.

2.5.1 Types

The following five types of descriptors are defined.

Device descriptor

This descriptor exists in every device and includes basic information such as the supported USB specification version, device class, protocol, maximum packet length that can be used when transferring data to endpoint 0, vendor ID, and product ID.

This descriptor is transmitted in response to a GET_DESCRIPTOR_Device request.

• Configuration descriptor

At least one configuration descriptor exists in every device and includes information such as the device attribute (power supply method) and power consumption. This descriptor is transmitted in response to a GET_DESCRIPTOR_Configuration request.

Interface descriptor

This descriptor is required for each interface and includes information such as the interface identification number, interface class, and supported number of endpoints. This descriptor is transmitted in response to a GET_DESCRIPTOR_Configuration request.

• Endpoint descriptor

This descriptor is required for each endpoint specified for an interface descriptor and defines the transfer type (direction), maximum packet length that can be used for a transfer, and transfer interval. However, endpoint 0 does not have this descriptor. This descriptor is transmitted in response to a GET DESCRIPTOR Configuration request.

• String descriptor

This descriptor includes any character string. This descriptor is transmitted in response to a GET_DESCRIPTOR_String request.

2.5.2 Format

The size and fields of each descriptor type vary as described below.

Remark The data sequence of each field is in little endian format.

Table 2-4 Device Descriptor Format

| Field | Size | Description | |
|--------------------|---------|--|--|
| | (Bytes) | | |
| bLength | 1 | Descriptor size | |
| bDescriptorType | 1 | Descriptor type | |
| bcdUSB | 2 | USB specification release number | |
| bDeviceClass | 1 | Class code | |
| bDeviceSubClass | 1 | Subclass code | |
| bDeviceProtocol | 1 | Protocol code | |
| bMaxPacketSize0 | 1 | Maximum packet size of endpoint 0 | |
| idVendor | 2 | Vendor ID | |
| idProduct | 2 | Product ID | |
| bcdDevice | 2 | Device release number | |
| iManufacturer | 1 | Index to the string descriptor representing the manufacturer | |
| iProduct | 1 | Index to the string descriptor representing the product | |
| iSerialNumber | 1 | Index to the string descriptor representing the device production number | |
| bNumConfigurations | 1 | Number of configurations | |

Remark Vendor ID: The identification number each company that develops a USB device acquires from USB-IF

Product ID: The identification number each company assigns to a product after acquiring the vendor ID

Table 2-5 Configuration Descriptor Format

| Field | Size (Bytes) | Description |
|---------------------|-----------------|---|
| bLength | 1 | Descriptor size |
| bDescriptorType | 1 | Descriptor type |
| wTotalLength | 2 | Total number of bytes of the configuration, interface, and endpoint |
| | | descriptors |
| bNumInterfaces | 1 | Number of interfaces in this configuration |
| bConfigurationValue | 1 | Identification number of this configuration |
| iConfiguration | 1 | Index to the string descriptor specifying the source code for this |
| | | configuration |
| bmAttributes | 1 | Features of this configuration |
| bMaxPower | 1 | Maximum current consumed in this configuration (in 2 μ A units) |

Table 2-6 Interface Descriptor Format

| Field | Size | Description | |
|--------------------|---------|--|--|
| | (Bytes) | | |
| bLength | 1 | Descriptor size | |
| bDescriptorType | 1 | Descriptor type | |
| bInterfaceNumber | 1 | Identification number of this interface | |
| bAlternateSetting | 1 | Whether the alternative settings are specified for this interface | |
| bNumEndpoints | 1 | Number of endpoints of this interface | |
| bInterfaceClass | 1 | Class code | |
| bInterfaceSubClass | 1 | Subclass code | |
| bInterfaceProtocol | 1 | Protocol code | |
| iInterface | 1 | Index to the string descriptor specifying the source code for this interface | |

Table 2-7 Endpoint Descriptor Format

| Field | Size (Bytes) | Description |
|------------------|-----------------|---|
| bLength | 1 | Descriptor size |
| bDescriptorType | 1 | Descriptor type |
| bEndpointAddress | 1 | Transfer direction of this endpoint Address of this endpoint |
| bmAttributes | 1 | Transfer type of this endpoint |
| wMaxPacketSize | 2 | Maximum packet size of this transfer |
| bInterval | 1 | Polling interval of this endpoint |

Table 2-8 String Descriptor Format

| Field | Size (Bytes) | Description | |
|-----------------|-----------------|-----------------|--|
| bLength | 1 | Descriptor size | |
| bDescriptorType | 1 | Descriptor type | |
| bString | Any | Any data string | |

CHAPTER 3 SAMPLE DRIVER SPECIFICATIONS

This chapter provides details about the features and processing of the USB Communication Device Class sample driver for the 78K0R/Kx3-L and the specifications of the functions provided in the 78K0R/Kx3-L.

3.1 Overview

3.1.1 Features

The sample driver can perform the following processing.

(1) Initialization

The USB function controller is set up for use by manipulating various registers. This setup includes specifying settings for the CPU registers of the 78K0R/Kx3-L and specifying settings for the registers of the USB function controller. For details, see **3. 2. 1 CPU Initialization**, **3. 2. 2 3. 2. 2 USB function controller initialization processing**

(2) Monitoring endpoints

The status of transfer endpoints in USB function controller is notified from INTUSB interrupt. There are CPUDEC interrupt expressing the request of decode by FW for the control transfer endpoint (Endpoint0) and BKO1DT interrupt showing the normal reception of data for bulk-out transfer (reception) endpoint (Endpoint2). During the processing of Endpoint0, requests are responded too. For details, see **3.2.3 INTUSB interrupt processing**.

(3) Sample application

The data at the endpoint for bulk-out transfer (reception) is read, and then the data is written to the endpoint for bulk-in transfer (transmission). For details, **see CHAPTER 4 SAMPLE APPLICATION SPECIFICATIONS.**

3. 1. 2 Supported requests

This section describes the USB requests supported by the sample driver.

(1) Standard requests

The sample driver returns the following responses for requests to which the 78K0R/Kx3-L does not automatically respond.

(a) **GET_DESCRIPTOR_string**

The host issues this request to acquire the string descriptor of the function device. If this request is received, the sample driver transmits the requested string descriptor to the host through a control read transfer.

(b) Other requests

The sample driver returns a STALL.

(2) Class requests

The sample driver responds to class requests of the CDC by using the following class requests.

(a) SendEncapsulatedCommand

This request is used to issue a command in the format of the CDC interface control protocol. If this request is received, the sample driver retrieves the data related to the request and then transmits them through bulk-in transfer.

(b) GetEncapsulatedResponse

This request is used to request a response in the format of the CDC interface control protocol. Currently, the sample driver does not support this request.

(c) SetLineCoding

This request is used to specify the serial communication format. If this request is received, the sample driver retrieves the data related to the request to specify settings such as the communication rate and then transmits a NULL packet through control read transfer.

(d) GetLineCoding

This request is used to acquire the current communication format settings on the device side. If this request is received, the sample driver reads settings such as the communication rate and then transmits them through control read transfer.

(e) SetControlLineState

This request is used for RS-232/V.24 format control signals. If this request is received the sample driver transmits a NULL packet through control read transfer.

3. 1. 3 Descriptor settings

The settings of each descriptor specified by the sample driver are shown below. These settings are included in header file "usbf78k0r_desc.h".

(1) Device descriptor

This descriptor is transmitted in response to a GET_DESCRIPTOR_device request. The settings are stored in the UF0DDn registers (where n = 0 to 17) when the USBF is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_device request.

| Field | Size | Specified | Description |
|--------------------|---------|-----------|--|
| | (Bytes) | Value | |
| bLength | 1 | 0x12 | Descriptor size: 18 bytes |
| bDescriptorType | 1 | 0x01 | Descriptor type: device |
| bcdUSB | 2 | 0x0200 | USB specification release number: USB 2.0 |
| bDeviceClass | 1 | 0x02 | Class code: CDC |
| bDeviceSubClass | 1 | 0x00 | Subclass code: none |
| bDeviceProtocol | 1 | 0x00 | Protocol code: No unique protocol is used |
| bMaxPacketSize0 | 1 | 0x40 | Maximum packet size of endpoint 0: 64 |
| idVendor | 2 | 0x0409 | Vendor ID:NEC |
| idProduct | 2 | 0x01CD | Product ID:78K0R /Kx3-L |
| bcdDevice | 2 | 0x0001 | Device release number:1st version |
| iManufacturer | 1 | 0x01 | Index to the string descriptor representing the manufacturer: 1 |
| iProduct | 1 | 0x02 | Index to the string descriptor representing the product: 2 |
| iSerialNumber | 1 | 0x03 | Index to the string descriptor representing the device production number:3 |
| bNumConfigurations | 1 | 0x01 | Number of configurations:1 |

Table 3-1 Device Descriptor Settings

(2) Configuration descriptor

This descriptor is transmitted in response to a GET_DESCRIPTOR_configuration request. The settings are stored in the UF0CIEn registers (where n = 0 to 255) when the USB function controller is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request.

| Field | Size (Bytes) | Specified Value | Description |
|---------------------|-----------------|--------------------|---|
| bLength | 1 | 0x09 | Descriptor size: 9 bytes |
| bDescriptorType | 1 | 0x02 | Descriptor type: configuration |
| wTotalLength | 2 | 0x0030 | Total number of bytes of the configuration, interface, and endpoint descriptors: 48 bytes |
| bNumInterfaces | 1 | 0x02 | Number of interfaces in this configuration: 2 |
| bConfigurationValue | 1 | 0x01 | Identification number of this configuration:1 |
| iConfiguration | 1 | 0x00 | Index to the string descriptor specifying the source code for this configuration:0 |
| bmAttributes | 1 | 0x80 | Features of this configuration: bus-powered, no remote wakeup |
| bMaxPower | 1 | 0x1B | Maximum current consumed in this configuration: 54 mA |

Table 3-2 Configuration Descriptor Settings

(3) Interface descriptor

This descriptor is transmitted in response to a GET_DESCRIPTOR_configuration request. The settings are stored in the UF0CIEn registers (where n = 0 to 255) when the USB function controller is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request.

Two types of descriptors are set up because the sample driver uses two interfaces.

Table 3-3 Interface Descriptor Settings for Interface 0

| Field | Size | Specified | Description |
|--------------------|---------|-----------|---|
| | (Bytes) | Value | |
| bLength | 1 | 0x09 | Descriptor size: 9 bytes |
| bDescriptorType | 1 | 0x04 | Descriptor type: interface |
| bInterfaceNumber | 1 | 0x00 | Identification number of this interface: 0 |
| bAlternateSetting | 1 | 0x00 | Whether the alternative settings are specified for this interface: no |
| bNumEndpoints | 1 | 0x01 | Number of endpoints of this interface: 1 |
| bInterfaceClass | 1 | 0x02 | Class code: communications interface class |
| bInterfaceSubClass | 1 | 0x02 | Subclass code: Abstract Control Model |
| bInterfaceProtocol | 1 | 0x00 | Protocol code: No unique protocol is used. |
| iInterface | 1 | 0x00 | Index to the string descriptor specifying the source code for this interface: |
| | | | 0 |

| Table 3-4 Interface Descriptor Settings for Interface 1 | | | | |
|---|---------|-----------|---|--|
| Field | Size | Specified | Description | |
| | (Bytes) | Value | | |
| bLength | 1 | 0x09 | Descriptor size: 9 bytes | |
| bDescriptorType | 1 | 0x04 | Descriptor type: interface | |
| bInterfaceNumber | 1 | 0x01 | Identification number of this interface: 1 | |
| bAlternateSetting | 1 | 0x00 | Whether the alternative settings are specified for this interface: no | |
| bNumEndpoints | 1 | 0x02 | Number of endpoints of this interface: 2 | |
| bInterfaceClass | 1 | 0x0A | Class code: communications interface class | |
| bInterfaceSubClass | 1 | 0x00 | Subclass code: Abstract Control Model | |
| bInterfaceProtocol | 1 | 0x00 | Protocol code: No unique protocol is used. | |
| iInterface | 1 | 0x00 | Index to the string descriptor specifying the source code for this interface: | |

0

Table 2.4 Interface Decorinter Settings for Interface 4

Endpoint descriptor (4)

This descriptor is transmitted in response to a GET_DESCRIPTOR_configuration request. The settings are stored in the UF0CIEn registers (where n = 0 to 255) when the USB function controller is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request. Three descriptor types are specified because the sample driver uses three endpoints.

| Field | Size | Specified | Description |
|------------------|---------|-----------|--|
| | (Bytes) | Value | |
| bLength | 1 | 0x07 | Descriptor size: 7 bytes |
| bDescriptorType | 1 | 0x05 | Descriptor type: endpoint |
| bEndpointAddress | 1 | 0x87 | Transfer direction of this endpoint: IN Address of this endpoint: 7 |
| bmAttributes | 1 | 0x03 | Transfer type of this endpoint: interrupt |
| wMaxPacketSize | 2 | 0x0008 | Maximum packet size of this transfer: 8 bytes |
| bInterval | 1 | 0x0A | Polling interval of this endpoint: 10 ms |

Table 3-5 Endpoint Descriptor Settings for Endpoint 7

Table 3-6 Endpoint Descriptor Settings for Endpoint 1

| Field | Size | Specified | Description | |
|------------------|---------|-----------|---|--|
| | (Bytes) | Value | | |
| bLength | 1 | 0x07 | Descriptor size: 7 bytes | |
| bDescriptorType | 1 | 0x05 | Descriptor type: endpoint | |
| bEndpointAddress | 1 | 0x81 | Transfer direction of this endpoint: OUT Address of this endpoint: 2 | |
| bmAttributes | 1 | 0x02 | Transfer type of this endpoint: bulk | |
| wMaxPacketSize | 2 | 0x0040 | Maximum packet size of this transfer: 64 bytes | |
| bInterval | 1 | 0x00 | Polling interval of this endpoint: 0 ms | |

Table 3-7 Endpoint Descriptor Settings for Endpoint 2

| Field | Size | Specified | Description | |
|------------------|---------|-----------|--|--|
| | (Bytes) | Value | | |
| bLength | 1 | 0x07 | Descriptor size: 7 bytes | |
| bDescriptorType | 1 | 0x05 | Descriptor type: endpoint | |
| bEndpointAddress | 1 | 0x02 | Transfer direction of this endpoint: IN Address of this endpoint: 2 | |
| bmAttributes | 1 | 0x02 | Transfer type of this endpoint: bulk | |
| wMaxPacketSize | 2 | 0x0040 | Maximum packet size of this transfer: 64 bytes | |
| bInterval | 1 | 0x00 | Polling interval of this endpoint: 0 ms | |

(5) String descriptor

This descriptor is transmitted in response to a GET_DESCRIPTOR_string request. If a GET_DESCRIPTOR_string request is received, the sample driver stores the settings of this descriptor into the UF0E0W register of the USB function controller.

Table 3-8 String Descriptor Settings

(a)String 0

| - | | | | |
|-----------------|---------|------------|-------------------------------|--|
| Field | Size | Specified | Description | |
| | (Bytes) | Value | | |
| bLength | 1 | 0x04 | Descriptor size: 4 bytes | |
| bDescriptorType | 1 | 0x03 | Descriptor type: string | |
| bString | 2 | 0x09, 0x04 | Language code: English (U.S.) | |

(b)String 1

| Field | Size | Specified | Description | |
|-----------------|---------|-----------|-------------------------------------|--|
| | (Bytes) | Value | | |
| bLength Note1 | 1 | 0x2A | Descriptor size: 42 bytes | |
| bDescriptorType | 1 | 0x03 | Descriptor type: string | |
| bString Note 2 | 40 | - | Vendor: NEC Electronics Corporation | |

Notes 1. The specified value depends on the size of the bString field.2. The vendor can freely set up the size and specified value of this field.

(c)String 2

| Field | Size | Specified | Description |
|-----------------|---------|-----------|-----------------------------------|
| | (Bytes) | Value | |
| bLength Note1 | 1 | 0x0E | Descriptor size: 14 bytes |
| bDescriptorType | 1 | 0x03 | Descriptor type: string |
| bString Note 2 | 12 | - | Product type: CDCDrv (CDC driver) |

Notes 1. The specified value depends on the size of the bString field.

2. The vendor can freely set up the size and specified value of this field.

(d)String 3

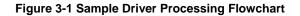
| Field | Size | Specified | Description | |
|-----------------|---------|-----------|---------------------------|--|
| | (Bytes) | Value | | |
| bLength Note1 | 1 | 0x16 | Descriptor size: 22 bytes | |
| bDescriptorType | 1 | 0x03 | Descriptor type: string | |
| bString Note2 | 20 | - | Serial number: 0_98765432 | |

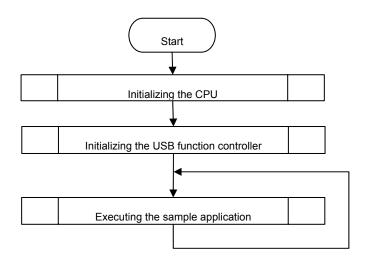
Notes1. The specified value depends on the size of the bString field

2. The vendor can freely set up the size and specified value of this field.

3.2 Operation of Each Section

The processing sequence below is performed when the sample driver is executed. This section describes each processing. For details about the sample application, see CHAPTER 4 SAMPLE APPLICATION SPECIFICATIONS.

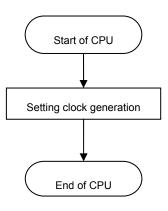




3. 2. 1 CPU Initialization

The settings necessary to use the USB function controller are specified.

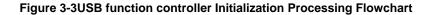
Figure 3-2 CPU Initialization Flowchart

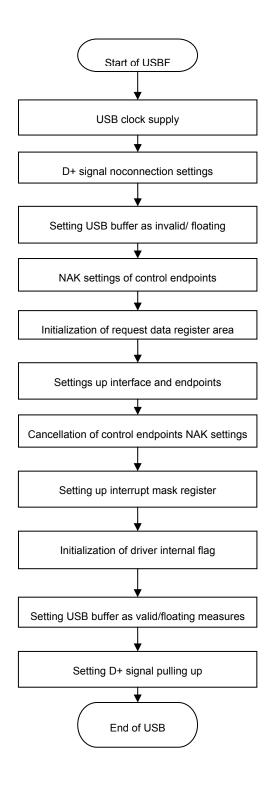


- Clock generation settings Operation of internal clock of CPU is set. Here, five registers are set.
 - (a) "0x41" is written to CMC register to specify X1 oscillation mode, $10MHz < f_{MX} <= 20MHz$.
 - (b) "0" is written to the MSTOP bit of CSC register to start the operation of X1 oscillation circuit.
 - (c) Oscillation stability time is verified according to OSTC register.
 - (d) "0x01" is written in PLLC register to stop the PLL operation.
 - (e) "0x38" is written to the CKC register to specify CPU/peripheral hardware clock to main system clock (f_{MAIN}), main system clock to high speed system clock (f_{MX}) and ratio of dividing frequency to f_{MX}.
 - (f) "1" is written to the HIPSTOP bit of CSC register to stop high speed built-in oscillation circuit.
 - (g) "1" is written to PLLM bit of PLLC register to multiply the frequency of the clock provided to PLL by 12.
 - (h) "0" is written to PLLSTOP bit of PLLC register to stat the operation of PLL.

3. 2. 2 USB function controller initialization processing

The settings necessary to use the USB function controller are specified.





(1) USB clock supply

"0x80" is set in UCKC register so that USB clock is supplied to USB function controller.

(2) D+ Signal no-connection settings

"0x02" is set to UF0GPR register in order to avoid being detected by the host.

(3) Invalidate USB buffer as and validate the floating measures

"0x00" is set to UF0BC register to disable the operations of USB function controller set as valid USB buffer and invalid floating measures.

(4) NAK settings of control endpoints

In order to avoide the unintended response before registering the data which are used for automatic respose by the hardware. 1 is written to the EP0NKA bit of the UF0E0NA register so that the hardware responds to all requests, including requests that are automatically responded to, with a NAK.

(5) Initializing the request data register area

The descriptor data transmitted in auto response to a GET_DESCRIPTOR request is added to the following registers.

- (a) 0x00 is written to the UF0DSTL register to disable remote wakeup and operate the USB function controller as a bus-powered device.
- (b) 0x00 is written to the UF0EnSL registers (where n = 0 to 2) to indicate that endpoint n operates normally.
- (c) The total data length (number of bytes) of the required descriptor is written to the UF0DSCL register to determine the range of the UF0CIEn registers (where n = 0 to 255).
- (d) The device descriptor data is written to the UF0DDn registers (where n = 0 to 7).
- (e) The data of the configuration, interface, and endpoint descriptors is written to the UF0CIEn registers (where n = 0 to 255).
- (f) 0x00 is written to the UF0MODC register to enable automatic responses to GET_DESCRIPTOR_configuration requests.
- (6) NAK settings of interface and endpoints

Information such as the number of supported interfaces, whether the alternative setting is used, and the relationship between the interfaces and endpoints are specified for various registers. The following registers are accessed.

- (a) 0x80 is written to the UF0AIFN register to enable two interfaces.
- (b) 0x00 is written to the UF0AAS register to disable the alternative setting.
- (c) 0x40 is written to the UF0E1IM register to link endpoint 1 to interface 1.
- (d) 0x40 is written to the UF0E2IM register to link endpoint 2 to interface 1.
- (e) 0x20 is written to the UF0E7IM register to link endpoint 7 to interface 0.
- (7) Disabling NAK settings of control endpoints

The NAK response operations for all requests are cancelled. 0 is written to the EP0NKA bit of the UF0E0NA register to restart responses corresponding to each request, including requests that are automatically responded to.

(8) Setting up the interrupt mask registers

Masking is specified for each USB function controller interrupt source. The following registers are accessed:

- (a) 0x00 is written to the UF0lcn registers (where n = 0 to 7) to clear all interrupt sources.
- (b) 0x00 is written to the UF0FICn registers (where n = 0 and 1) to clear all transfer FIFOs.
- (c) 0x7B is written to the UF0IM0 register to mask all interrupt sources other than BUSRST interrupt and SETRQ interrupt from the interrupt sources indicated by the UF0IS0 register.
- (d) 0x7E is written to the UF0IM1 register to mask all interrupt sources other than CPUDEC interrupt from the interrupt sources indicated by the UF0IS1 register.
- (e) 0xF3 is written to the UF0IM2 register to mask all interrupt sources indicated by the UF0IS2 register.
- (f) 0xFE is written to the UF0IM3 register to mask interrupt sources indicated by the UF0IS3 register other than those of the BKO1DT interrupt.
- (g) 0xFF is written to the UF0IM4 register to mask all interrupt sources indicated by the UF0IS4 register.
- (h) "0" is written to the USBIF bit of CPU to clear INTUSB interrupt.
- (i) "0" is written to the USBMK bit of CPU to disable mask of INTUSB interrupt.

(9) Initialization of driver internal flag

A high level signal is output from the D+ pin to report to the host that a device has been connected. For the sample driver, the connections shown in Figure 3-4 are assumed and the following registers are accessed.

(10)USB buffer enabled/ floating measures disabled

"0x03" is set to UF0BC register to enable USB buffer, to disable floating measures and to enable USB

function controller operations.

(11)Pulling up the D+ signal

"0x02" is set to UF0GPR register to report to the host that a device has been connected.

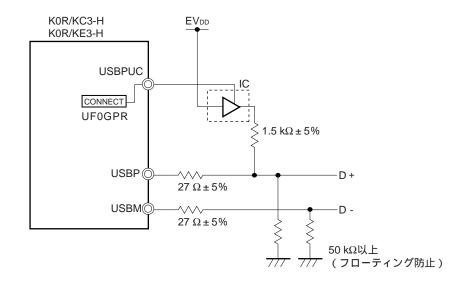
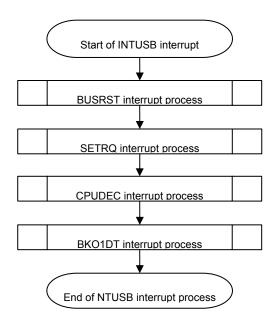


Figure 3-4 USB function controller Connection Example

3. 2. 3 INTUSB interrupt process

Interrupt request (INTUSB) from USB function controller reports only about the interrupts which are masked. Disable mask at the initialization for the necessary interrupts. Respective necessary processes are executed for the reported interrupts.

Figure 3-5 Process flow of Endpoint0 monitoring



(1) BUSRST interrupt process

It is reports when Bus Reset is generated.

Process is executed in the following order.

- (a) "0x7F" is written to the UF0IC0 to clear BUSRST interrupt.
- (b) "1" is written to usbf78k0r_busrst_flg flag.
- (c) usbf78k0r_buff_init () function is called.

(2) SETRQ interrupt process

SET_XXXX request for auto process is received and it is reported at auto processing.

Process is executed in the following order.

- (a) "0xFB" is written to the UF0IC0 to clear SETRQ interrupt.
- (b) Both SETCON bit of UF0SET register and CONF bit of UF0MODS register are set to "1" is verified. "1" is set to CONFIGURATION by the SET_CONFIGURATION request is indicated.
- (c) "0" is written to the usbf78k0r_busrst_flg flag to report that it is switched from reset state to normal state.
- (3) CPUDEC interrupt process

It is reported when FW process request is received.

Process is executed in the following order.

- (a) "0xFD" is written to UF0IC1 register to clear PROT interrupt.
- (b) UF0E0ST register is read for 8 times then request data is acquired and decoded.
- (c) If request is class request, usbf78k0r_classreq () function is called and class request process is executed.
- (d) If request is not class request, usbf78k0r_standardreq () function is called and standard request process is executed.
- (4) BKO1DT interrupt process

It is reported when data is received in UF0BO1 register normally.

Process is executed in the following order.

- (a) "0xFE" is written to the UF0IC3 register to clear BKO1DT interrupt.
- (b) "1" is set to (usbf78k0r_rdata_flg) flag indicating existence of received data to indicate that there is received data in bulk out endpoint in the drive. This flag is originally defined by the sample driver.

3.3 Function Specifications

This section describes the functions implemented in the sample driver.

3.3.1 Functions

The functions of each source file included in the sample driver are described below.

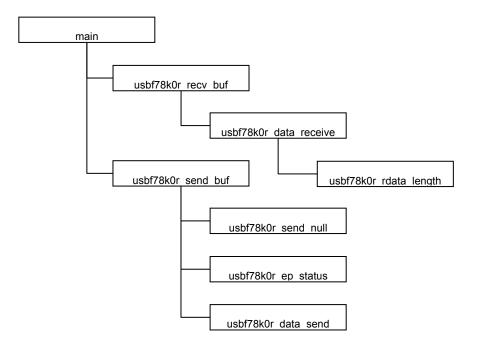
Table 3-9 Functions in the Sample Driver

| Source File | Function Name | Description |
|---------------------------|-------------------------------------|---|
| main.c | cpu_init | Initializes the CPU. |
| | main | Main routine |
| usbf78k0r.c | usbf78k0r_init | Initializes the USB function controller |
| | usbf78k0r_intusbf0 | Processing INTUSB interrupt |
| | usbf78k0r_standardreq | Processes standard requests. |
| | usbf78k0r_getdesc | Processes GET_DESCRIPTOR(String) |
| | usbf78k0r_send_EP0 | Transmits Endpoint0 |
| | usbf78k0r_receive_EP0 | Receives Endpoint0 |
| | usbf78k0r_sendnullEP0 | Transmits a NULL packet for endpoint 0. |
| | usbf78k0r_sendstallEP0 | Transmit a STALL for endpoint 0. |
| | usbf78k0r_ep_status | Notifies FIFO status of bulk/interrupt Inn end point |
| | usbf78k0r_send_null | Transmits a NULL packet of bulk/interrupt inn endpoint |
| | usbf78k0r_data_send | Transmits bulk/interrupt Inn end point |
| | usbf78k0r_rdata_length | Acquires the bulk out endpoint received data length |
| | usbf78k0r_data_receive | Receives bulk out endpoint |
| | usbf78k0r_fifo_clear | Clears bulk/interrupt Inn end point and bulk out endpoint FIFO |
| usbf78k0r_communication.c | usbf78k0r_classreq | Processes CDC class/request |
| | usbf78k0r_send_encapsulated_command | Processes SendEncapsulatedCommand requests |
| | usbf78k0r_get_encapsulated_response | Processes Get Encapsulated Response requests |
| | usbf78k0r_set_line_coding | Processes SetLineCoding requests. |
| | usbf78k0r_get_line_coding | Processes GetLineCoding requests. |
| | usbf78k0r_set_control_line_state | Processes SetControlLineState requests. |
| | usbf78k0r_buff_init | Clears FIFO of endpoint for CDC data transfer |
| | usbf78k0r_get_bufinit_flg | Notifies execution state of FIFO initialization process |
| | usbf78k0r_send_buf | Transmits CDC data |
| | usbf78k0r_recv_buf | Receives CDC data |

3. 3. 2 Correlation of the functions

Some functions call other functions during the processing. The following figures show the correlation of the functions.







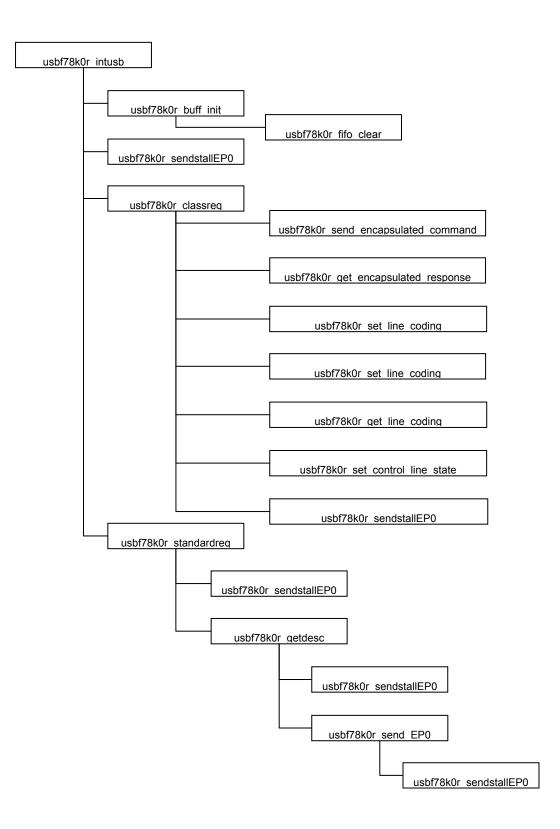


Figure 3-8 Calling Functions during the Processing for the USB Communication Class (1)

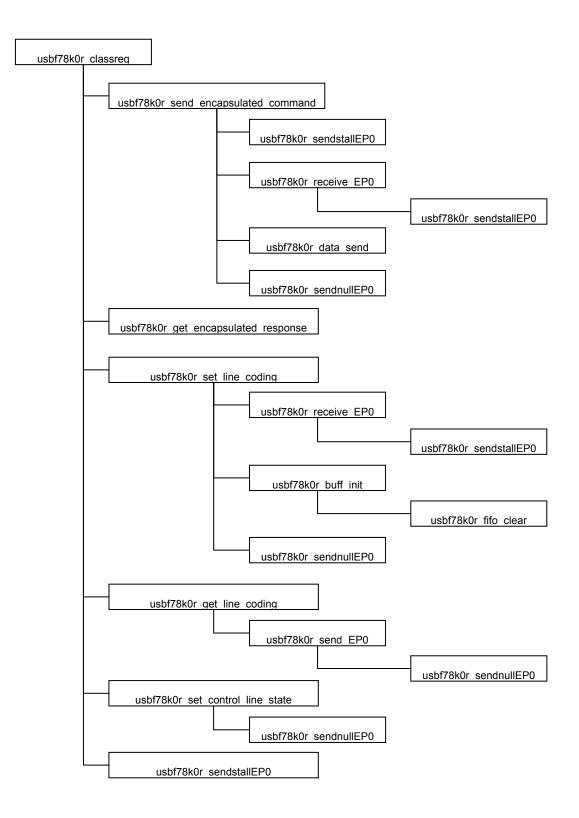
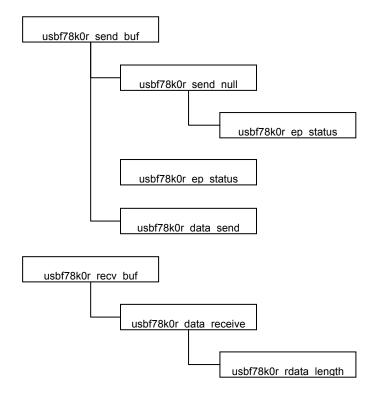


Figure 3-9 Calling Functions during the Processing for the USB Communication Class (2)



3.3.3 Function features

This section describes the features of the functions implemented in the sample driver.

(1) Function description format

The functions are described in the following format.

Function name

[Overview]

An overview of the function is provided

[C description format]

The format in which the function is written in C is provided.

[Parameters]

The parameters (arguments) of the function are described.

| Parameter | Description |
|--------------------|-------------------|
| Parameter type and | Parameter summary |
| name | |

[Return values]

The values returned by the function are described.

| Symbol | Description |
|-------------------|----------------------|
| Return value type | Return value summary |
| and name | |

[Description]

The feature of the function is described

Functions for the main routine

main

[Overview] Main processing

[C description format] void main(void)

[Parameters] None

[Return value] None

[Description]

This function is called first when the sample driver is executed. This function calls the initialization function of CPU, initialization function of USB function controller and then the sample application processing function sequentially.

cpu_init

[Overview] Initializes the CPU.

[C description format] void cpu_init(void)

[Parameters] None

[Return value] None

[Description]

This function is called in the main processing. The settings those are necessary to use the USB function controller in the 78K0R/Kx3, such as the clock frequency, and operation mode.

Functions for the USB function controller

usbf78k0r_init

[Overview] Initializes the USB function controller

[C description format] void usbf78k0r_init(void)

[Parameters] None

[Return value] None

[Description]

This function is called during initialization processing.

This function specifies the settings required for using the USBF, such as allocating and

specifying the data area and masking interrupt requests.

usbf78k0r_intusbf0

[Overview]

INTUSB interrupt processing

[C description format]

__interrupt void usbf78k0r_intusbf0 (void)

[Parameters] None

[Return value] None

[Description]

This function is an interrupt service routine called from INTUSBF0 interrupt.

Generated interrupt processing is done while verifying about the interrupt requests about the interrupt which are not masked of USB function controller.

usbf78k0r_standardreq

[Overview]

Processes standard requests to which the USB function controller does not automatically respond

[C description format]

void usbf78k0r_standardreq (USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value] None

[Description]

This function is called from the CPUDEC interrupt cause process of INTUSB interrupt process.

If a GET_DESCRIPTOR request is decoded, this function calls the GET_DESCRIPTOR request processing function (usbf78k0r_getdesc). For other requests, this function calls the function for returning STALL responses for endpoint 0 (usbf78k0r_sendstallEP0).

usbf78k0r_getdesc

[Overview]

Processes GET_DESCRIPTOR requests

[C description format]

void usbf78k0r_getdesc (USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value] None

[Description]

This function is called during the processing of standard requests to which the USB function controller does not automatically respond. If a decoded request requests a string descriptor, this function calls the USB data transmission function (usbf78k0r_send_EP0) for endpoint 0 and transmits a string descriptor from endpoint 0. If a decoded request requests any other descriptor, this function calls the function for processing STALL responses (usbf78k0r_sendstallEP0) for endpoint 0.

usbf78k0r_send_EP0

[Overview]

Transmits USB data for Endpoint0

[C description format]

INT32 usbf78k0r_send_EP0(UINT8* data, INT32 len)

[Parameters]

| Parameter | Description |
|-------------|----------------------------------|
| UINT8* data | Transmission data buffer pointer |
| INT32 len | Transmission data length |

[Return value]

| Symbol | Description |
|-----------|----------------------|
| DEV_OK | Normal completion |
| DEV_ERROR | Abnormal termination |

[Description]

This function stores the data stored in the transmission data buffer into the FIFO for the specified Endpoint0, byte by byte.

usbf78k0r_receive_EP0

[Overview]

Receives USB data for Endpoint0

[C description format]

INT32 usbf78k0r_receive_EP0(UINT8* data, INT32 len)

[Parameters]

| Parameter | Description |
|-------------|-------------------------------|
| UINT8* data | Reception data buffer pointer |
| INT32 len | Reception data length |

[Return value]

| Symbol | Description |
|-----------|----------------------|
| DEV_OK | Normal completion |
| DEV_ERROR | Abnormal termination |

[Description]

This function reads data from the FIFO for the specified endpoint byte by byte and stores the data into the reception data buffer.

usbf78k0r_sendnullEP0

[Overview] Transmits a NULL packet for endpoint 0

[C description format] void usbf78k0r_sendnullEP0(void)

[Parameters] None

[Return value] None

[Description]

This function clears the FIFO for endpoint 0 and transmits a NULL packet from the USBF

by setting the bit that indicates the end of data to 1.

usbf78k0r_sendstallEP0

[Overview]

Returns a STALL for endpoint 0

[C description format]

void usbf78k0r_sendstallEP0(void)

[Parameters] None

[Return value] None

[Description]

This function makes the USBF return a STALL by setting the bit that indicates the use of STALL for Endpoint 0 to 1.

usbf78k0r_ep_status

[Overview]

Notifies FIFO status for bulk/interrupt inn endpoint

[C description format] INT32 usbf78k0r_ep_status(INT8 ep)

[Parameters]

| | Parameter | Description |
|---|-----------|-----------------------------------|
| INT8 ep Data transmission endpoint number | ер | Data transmission endpoint number |

[Return value]

| Symbol | Description |
|-----------|----------------------------------|
| DEV_OK | Normal completion (FIFO empty) |
| DEV_ERROR | Abnormal termination (FIFO full) |
| DEV_RESET | During Bus Reset processing |

[Description]

This function notifies the FIFO status of specified endpoint (for transmission).

usbf78k0r_send_null

[Overview]

Transmits a NULL packet for bulk/interrupt inn endpoint

[C description format]

INT32 usbf78k0r_send_null(INT8 ep)

[Parameters]

| Parameter | Description |
|-----------|------------------------------------|
| INT8 ep | Data transmission end point number |

[Return value]

| Valuo] | |
|-----------|----------------------|
| Symbol | Description |
| DEV_OK | Normal completion |
| DEV_ERROR | Abnormal termination |

[Description]

This function transmits a NULL packet from USB function controller by clearing the FIFO of specified Endpoint (for transmission) and setting the bit that indicates the end of data to 1.

usbf78k0r_data _send

[Overview]

Transmits USB data for bulk/interrupt Inn end point

[C description format]

INT32 usbf78k0r_data_send(UINT8* data, INT32 len, INT8 ep)

[Parameters]

| Parameter | Description |
|-------------|------------------------------------|
| UINT8* data | Transmission data buffer pointer |
| INT32 len | Transmission data length |
| INT8 ep | Data transmission end point number |

[Return value]

| Symbol | Description |
|------------|-------------------------------|
| len (>= 0) | Normal transmission data size |
| DEV_ERROR | Abnormal termination |

[Description]

This function stores the data stored in the transmission data buffer into the FIFO for the specified endpoint, byte by byte.

usbf78k0r_rdata _length

[Overview]

Acquires the USB reception data length

[C description format]

void usbf78k0r_rdata_length(INT32 *len , INT8 ep)

[Parameters]

| Parameter | Description |
|------------|---|
| INT32* len | Pointer to the storage address of the received data |
| | length |
| INT8 ep | Data reception endpoint number |

[Return value]

None

[Description]

This function reads the received data length of the specified endpoint. (For reception).

usbf78k0r_data _receive

[Overview]

Receives USB data for bulk end point

[C description format]

INT32 usbf78k0r_data_receive(UINT8* data, INT32 len, INT8 ep)

[Parameters]

| Parameter | Description |
|-------------|--------------------------------|
| UINT8* data | Reception data buffer pointer |
| INT32 len | Reception data length |
| INT8 ep | Data reception endpoint number |

[Return value]

| Symbol | Description |
|--------------|-------------------------------|
| len (>= 0) | Normal transmission data size |
| DEV_ERROR | Abnormal termination |

[Description]

This function reads data from the FIFO for the specified endpoint byte by byte and stores the data into the reception data buffer.

usbf78k0r_fifo_clear

[Overview]

Clears the FIFO for bulk/interrupt Endpoint

[C description format]

void usbf78k0r_fifo_clear(INT8 in_ep, INT8 out_ep)

[Parameters]

| Parameter | Description |
|-------------|------------------------------------|
| INT8 in_ep | Data transmission end point number |
| INT8 out_ep | Data reception end point number |

[Return value]

None

[Description]

This function clears the FIFO of Endpoint specified in bulk/interrupt Endpoint and clears (0) data reception flag (usbf78k0r_rdata_flg).

Functions for USB communication device class processing

usbf78k0r_classreq

[Overview]

Processes class request

[C description format]

void usbf78k0r_classreq(USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value]

None

[Description]

This function is called from the CPUDEC interrupt cause process of INTUSB interrupt process.

If a decoded request is communication class request, this function calls the each request processing function. For other requests, this function calls the function for returning a STALL for Endpoint0 (usbf78k0r_sendstallEP0).

usbf78k0r_send_encapsulated_command

[Overview]

Processes SendEncapsulatedCommand requests

[C description format]

void usbf78k0r_send_encapsulated_command(USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value] None

[Description]

If request decoded in the class request process is Send Encapsulated Command, this function is called. This function calls the data reception function (usbf78k0r_receive_EP0) to retrieve the data received at endpoint 0, and then calls the data transmission function (usbf78k0r_data_send) to transmit data from endpoint 2 via bulk-in transfer (transmission) and calls the NULL packet transmission function (usbf78k0r_sendnullEP0) for Endpoint0.

usbf78k0r_set_line_coding

[Overview]

Processes SetLineCoding requests

[C description format]

void usbf78k0r_set_line_coding(USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value]

None

[Description]

This function is called if request decoded at class request process is Set Line Coding. This function calls the data reception function (usbf78k0r_receive_EP0) to retrieve the data received at endpoint 0, and then writes the data to the UART_MODE_INFO structure. This function calls the FIFO initialization function (usbf78k0r_buff_init) for user data and then calls the NULL packet transmission function for endpoint 0 (usbf78k0r_sendnullEP0).

usbf78k0r_get_control_line_coding

[Overview]

Processes GetLineCoding requests

[C description format]

void usbf78k0r_get_line_coding(USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value] None

[Description]

This function is called if request decoded at class request process is Get Line Coding. This function transmits the UART_MODE_INFO structure value from Endpoint0 by calling USB data transmission function (usbf78k0r_send_EP0) for Endpoint0.

usbf78k0r_set_control_line_state

[Overview]

Processes SetControlLineState requests.

[C description format]

void usbf78k0r_set_control_line_state(USB_SETUP *req_data)

[Parameters]

| Parameter | Description |
|---------------------|--------------------------------------|
| USB_SETUP *req_data | Request data storage pointer address |

[Return value] None

None

[Description]

This function is called if request decoded in the class request process is "Set Control Line State". This function calls the NULL packet transmission function for endpoint 0 (usbf78k0r_sendnullEP0).

usbf78k0r_buff_init

[Overview]

Initializes the FIFO for user data

[C description format] void usbf78k0r_buff_init(void)

[Parameters] None

[Return value] None

[Description]

This function initializes the FIFO for communication class user data by calling FIFO clear function (usbf78k0r_fifo_clear) for bulk/interrupt Endpoint and sets the flag (usbf78k0r_bufinit_flg) that indicates transmission packet size of internal driver as clear (0) and FIFO initialization to 1.

usbf78k0r_get_bufinit_flg

[Overview]

Notifies FIFO status for user data

[C description format]

INT32 usbf78k0r_get_bufinit_flg(void)

[Parameters] None

[Return_value]

| Symbol | Description |
|-----------|----------------------------|
| DEV_OK | Normal status |
| DEV_ERROR | FIFO initialization status |

[Description]

This function notifies the internal driver flag (usbf78k0r_bufinit_flg) status that indicates the initialization of FIFO. If flag is set as 1, it indicates that FIFO is initialized and then it notifies the initialization status and clears flag to 0.

usbf78k0r_send_buf

[Overview]

Transmits user data for communication class

[C description format]

INT32 usbf78k0r_send_buf(UINT8* data, INT32 len)

[Parameters]

| Description |
|----------------------------------|
| Transmission data buffer pointer |
| Transmission data length |
| |

[Return_value]

| Symbol | Description |
|--------------|---------------------------------|
| len (>= 0) | Normal transmission data length |
| DEV_ERROR | Abnormal termination |

[Description]

This function transmits NULL packet that calls the NULL packet transmission function (usbf78k0r_send_null) for bulk/interrupt inn Endpoint, if transmission data size (Parameter:len) is 0 and size of the packet transmitted earlier (g_send_size) is Max Packet Size. If transmission data size (Parameter:len) is greater than 0 and transmission FIFO has null status (return value of usbf78k0r_ep_status is DEV_OK), this function calls the USB data transmission function (usbf78k0r_data_send). If data transmission is completed normally, it stores the size of the data transmitted to transmission completion packet size (g_send_size) defined in the driver.

usbf78k0r_recv_buf

[Overview]

Receives user data for communication class

[C description format]

INT32 usbf78k0r_recv_buf(UINT8* data, INT32 len)

[Parameters]

| Parameter | Description |
|-------------|-------------------------------|
| UINT8* data | Reception data buffer pointer |
| INT32 len | Reception data length |

[Return value]

| Symbol | Description |
|--------------|---------------------------------|
| len (>= 0) | Normal transmission data length |
| DEV_ERROR | Abnormal termination |

[Description]

This function calls USB data reception function (usbf78k0r_data_receive).

CHAPTER 4 SAMPLE APPLICATION SPECIFICATIONS

This chapter describes the sample application included with the sample driver.

4.1 Overview

The sample application is provided as a simple example of using the USB communication device class driver and is incorporated in the main routine of the sample driver.

The sample application reads the data received by the USB function controller and then transmits the read data. Various functions of the sample driver are used during this processing.

4.2 Operation

The sample application performs the processing shown in the following flowchart.

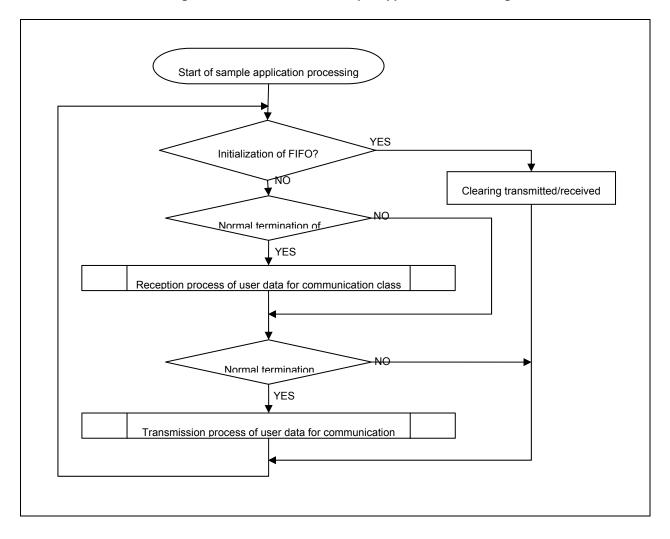


Figure 4-1 Flowchart for the Sample Application Processing

(1) Verifying FIFO initialization for user data

FIFO status notification function (usbf78k0r_get_bufinit_flg) for user data is called and if it is in normal state, verification process of transmission processing result is executed and if it is in the initialization state, transmission/reception result clear process (clearing transmission/reception process result of user data for communication class to 0) is executed.

(2) Verifying transmission process result of user data for communication class

If transmission process result of user data for communication class is Normal completion (and initial state), control shifts over to reception process of user data for communication class and if it is abnormal termination state, shifts to reception process result confirmation process.

(3) Reception process of user data for communication class

Buffer address, buffer size storing reception data is specified and reception function (usbf78k0r_recv_buf) of user data for communication class is called.

(4) Verifying reception process result of user data for communication class

If reception process result of user data for communication class is Normal completion (and initial state), control shifts over to transmission process of user data for communication class and if it is abnormal termination state, shifts to FIFO initialization confirmation process for user data.

(5) Transmission process of user data for communication class

Buffer size where data to be transmitted is stored and transmission data size are specified and transmission function (usbf78k0r_send_buf) of user data for communication class is called.

4.3 Using Functions

The main.c source file that includes this sample application is coded as follows in order to call sample driver functions. For details about the functions, see **3.3** Specifications of Functions.

(1) Definitions and declarations

2 header files "usbf78k0r.h" and "usbf78k0r_communication.h" are included in order to use the sample driver functions. User buffer (UserBuf) of size sufficient to process the 1 packet data for user data is set. (Maximum packet size of bulk endpoint in Full Speed USB is set to 64Byte)

(2) Initialization processing of CPU

Initialization processing of CPU function (cpu_init) is called.

- (3) Initialization process of USB function controller
 USB function controller initialization function (usbf78k0r_init) is called.
- (4) Verification of FIFO status for user data

FIFO state notification function (usbf78k0r_get_bufinit_flg) for user data is called and FIFO status is verified.

(5) Reception process of user data

User data reception function (usbf78k0r_recv_buf) for communication class is called and result is stored.

(6) Transmitting user data

User data transmission function (usbf78k0r_send_buf) for communication class is called and result is stored.

(7) Clearing process of transmission/reception process result

If FIFO for user data is initialized, transmission/reception process result stored in (5), (6) is cleared to 0.

```
void main(void)
        {
            INT32 rcv_ret = 0;
            INT32 snd_ret = 0;
            cpu_init();
            DI();
            usbf78k0r_init();
                                   /* initial setting of the USB Function */
            EI();
            while(1)
            {
                if (usbf78k0r get bufinit flg() != DEV ERROR) {
                    if (snd_ret >= 0) {
                       rcv_ret = usbf78k0r_recv_buf(&UserBuf[0], USERBUF_SIZE);
                    }
                    if (rcv_ret >= 0) {
                        snd_ret = usbf78k0r_send_buf(&UserBuf[0], rcv_ret);
                    }
               }
                else {
                    snd_ret = 0;
                    rcv_ret = 0;
               }
            }
       }
```

List 4-1 Sample Application Code (Portion)

CHAPTER 5 DEVELOPMENT ENVIRONMENT

This chapter provides an example of creating an environment for developing an application program that uses the USB communication device class sample driver for the 78K0R/Kx3-L and the procedure for debugging the application.

5.1 Development environment

This section describes the used hardware and software tool products.

5.1.1 Program development

The following hardware and software are necessary to develop a system that uses the sample driver.

| Components | | Product Example | Remark | |
|------------|-----------------------------|-----------------|---|--|
| Hardware | Host machine | - | PC/AT compatible computer (OS : Windows | |
| | | | XP) | |
| Software | Integrated development tool | PM+ | V6.31 | |
| | Compiler | CC78K0R | W2.12 | |
| | Assembler | RA78K0R | W1.33 | |

Table 5-1 Example of the Components Used in a Program Development Environment

5.1.2 Debugging

The following hardware and software are necessary to debug a system that uses the sample driver.

Table 5-2 Example of the Components Used in a Debugging Environment

| Components | | Product Example | Remark |
|------------|-----------------------------|-------------------|--|
| Hardware | Host machine - | | PC/AT compatible computer (OS : Windows XP) |
| | Target device | TK-78K0R/KE3L+USB | |
| | Inn circuit emulator | MINICUBE2 | |
| | USB cables | - | miniB-to-A connector cable |
| Software | Integrated development tool | PM+ | V6.31 |
| | Debugger | ID78K0R-QB | V3.60 |
| Files | Device file | DF78102664.78K | For the 78K0R/Kx3-L |
| | Project files | - | Note1 |

Notes 1.For details about products and how to obtain them, contact NEC Electronics.

2. A file that is used when creating a system using PM+ is included with the sample driver.

5. 2 Setting up the Environment

This section describes the preparations required for developing and debugging a system by using the products described in **5.1 Development environment.**

5. 2. 1 Preparing the host environment

Create a dedicated workspace on the host for debugging.

Installing an integrated development tool

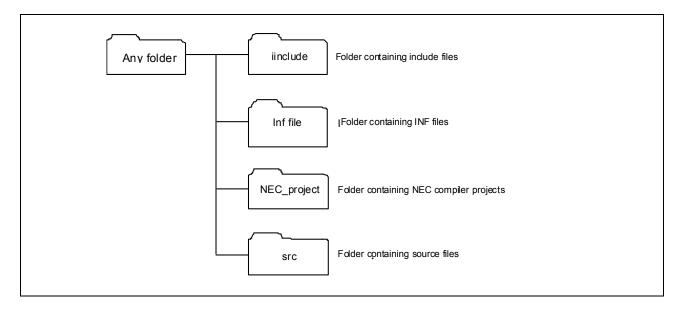
Install PM+. For details, see the PM+ User's Manual.

Downloading drivers

Store the set of files provided with the sample driver in any directory without changing the folder structure.

Store the device driver in any directory.

Figure 5-1 Folder Structure of the Sample Driver



Setting up the workspace

The procedure for using project files included with the sample driver is described below.

<1> Start PM+, and then select "**Open Workspace**" in the "**File**" menu.

| 器 P | M+ - | No W | /orksp | ace [| Project | Windo | w] |
|---------|-------------------------------|-------------------|--------|-------|------------------|-------|------|
| File | Edit | Find | Layer | View | Project | Build | Tool |
| 0 Ir | ew pen Isert fi lose | le | | | Ctrl+N Ctrl+O | | 2 |
| N | ew Wo | orkspac | :e | | | | |
| 0 | pen W | orkspa | ice | | | | |
| | | orkspa 'orkspa | | | | | |

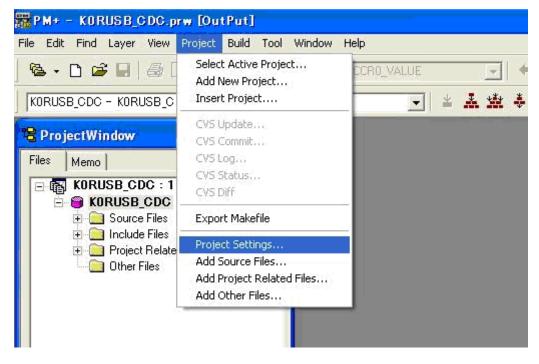
<2> In the **Open Workspace** dialog box, specify the workspace file in the NEC_project folder, which is the sample driver installation directory.

| Open Works | pace | ? 🔀 |
|------------------------|-----------------------|--------------|
| Look jn: 🔀 | NEC_project | * 📰 - |
| KORUSB | DDC.prw | |
| | | |
| | | |
| | | |
| | | |
| < | | > |
| File <u>n</u> ame: | KORUSB_CDC.prw | <u>O</u> pen |
| Files of <u>t</u> ype: | Workspace File(*.prw) | Cancel |
| | | Help |
| 1 | | |

Installing a device file

The procedure for using a device file for the 78K0R/Kx3-L is described below.

<1> Select **Project Settings** in the PM+ **Project** menu.



<2> In the **Project Settings** dialog box, click the **Device Install** button on the **Project Information** tab to start the Device File Installer.

| Project Settings | | | | |
|---------------------------|---------------------------|------------------|---------|----------------|
| Project Information Sourc | e File Tool Version Set | tings | | |
| Project File Name : | KORUSB_CDC.prj | | | |
| Folder : | D:\TK-KOR_USB\NE | C_project | | |
| Workspace File Name | : D:\TK-KOR_USB\NE | C project\KORUSB | CDC.prw | |
| Project Group : | | | | |
| Project <u>T</u> itle : | | | | |
| KORUSB_CDC | | | | |
| Microcontrollers Name | | Device Name : | | |
| 78K0R | • | uPD78F1026_64 | • | Device Install |
| | | | | |
| | | | | |
| | | | | Cancel Help |

<3> In the **Device File Installer** dialog box, click the **Install** button to start the installation wizard.

| evice File Installer | | | | |
|--|--|--|--|---------------|
| Device File Package | | I | | <u>H</u> elp |
| | | | | <u>A</u> bout |
| Device File | 15 | | | Get New Softw |
| Source <u>D</u> irectory Sel | ect: | | ✓ Browse | |
| Device Name | Version | Series | File Name | - |
| | | | | |
| Move Regis | ter <u>U</u> nRe | | Delete <u>Fi</u> le | |
| Move Regis | ter <u>U</u> nRe | gister | Delete <u>Ele</u> <u>Change registered directory</u> | |
| Move Regis Registry Device Name | ter <u>U</u> nRe | gister | Delete <u>File</u> <u>Change registered directory Directory </u> | |
| Move Regist Registry Device Name uPD78F0730 uPD78F1012_80 | ter UnRe Version V1.10 E1.00c | gister Series 78K0 78K0R | Delete Ele <u>Change registered directory</u> Directory C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I | |
| Move Registry Registry Device Name uPD78F0730 uPD78F1012_80 uPD78F1026_64 | ter UnRe Version V1.10 E1.00c E1.00c | gister Series 78K0 78K0R 78K0R 78K0R | Delete Ele Change registered directory Directory C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I | |
| Move Regist Registry Device Name uPD78F0730 uPD78F1012_80 uPD78F1026_64 uPD78F1022_48 uPD78F1023_48 | ter UnRe Version V1.10 E1.00c E1.00c E1.00c E1.00c E1.00c | gister Series 78K0 78K0R 78K0R 78K0R 78K0R 78K0R 78K0R | Delete Eile Change registered directory Directory C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I | |
| Move Regist Registry Device Name uPD78F0730 uPD78F1012_80 uPD78F1026_64 uPD78F1022_48 uPD78F1023_48 uPD78F1023_48 uPD78F1024_48 | ter UnRe Version V1.10 E1.00c E1.00c E1.00c E1.00c E1.00c E1.00c | gister Series 78K0 78K0R 78K0R 78K0R 78K0R 78K0R 78K0R | | |
| Move Regist Registry Device Name uPD78F0730 uPD78F1012_80 uPD78F1026_64 uPD78F1022_48 uPD78F1023_48 | ter UnRe Version V1.10 E1.00c E1.00c E1.00c E1.00c E1.00c | gister Series 78K0 78K0R 78K0R 78K0R 78K0R 78K0R 78K0R | Delete Eile Change registered directory Directory C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I C:\PROGRAM FILES\NEC I | Exit |

<4> In the Install Information File dialog box, click the Browse button.

| Install Wizard : Install Information File | \mathbf{X} |
|--|--------------|
| Installing from Device File Product Disk. Open a drive or directory which the Device File Product Disk exists, and select the Install-Information-File (CSETUP.INI or NECSETUP.INI).]] B <u>r</u> owse | |
| A: <u>F</u> D Browse | |
| < <u>B</u> ack. <u>N</u> ext > Cancel | |

- <5> In the Open dialog box, open the directory in which the device file was stored, select "NECSETUP.INI", file and then click the **Open** button. <6> In the message about usage permission, click the **Next** button.
- <7> In the File type selection dialog box, click of Next button after selecting relevant device files.

| Install Wizard : Kind of File | × |
|--|---|
| | |
| Select the kind of installation file : | |
| ▼ [78K0R/KC3-L(USB) | |
| ✓ 78K0R/KE3-L(USB) | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| < <u>B</u> ack <u>N</u> ext > Cancel | |

* Since screen is under development it can differ with the actuals

<8> [In the **Install Directory** dialog box, confirm that a path is displayed, and then click the **Next** button.

| Install Wizard : Install Directory | \mathbf{X} |
|---|--------------|
| | |
| Input the destination directory of the Common Device File : | |
| C:\Program Files\NEC Electronics Tools\DEV | |
| | |
| | |
| | |
| | |
| | |
| | |
| < <u>B</u> ack <u>N</u> ext > Cancel | |

<9> In the Installation Start dialog box, click the Next button.

| stall Wizard : Installation Start | × |
|--|----|
| | |
| When you select [Next], installation will start as below conditions. | |
| | |
| Installation files and destination directories : | |
| 78K0R/KC3-L(USB) d:\program files\nec electronics tools\dev | |
| 78K0R/KE3-L(USB) d:\program files\nec electronics tools\dev | |
| | |
| | |
| J | |
| | |
| | |
| < <u>B</u> ack <u>Next</u> Canc | el |

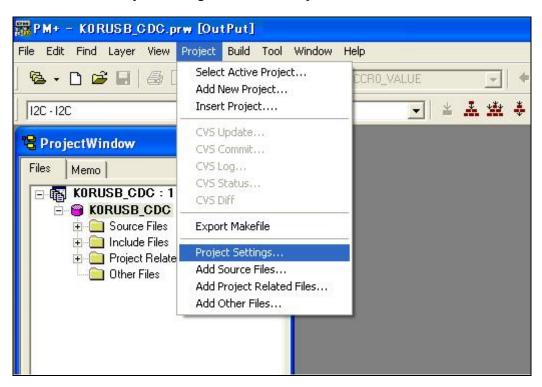
* Since screen is under development it can differ with the actuals.

- <10> The device file is installed to the project. This might take a while depending on the environment.
- <11> In the **Installation Finished** dialog box, click the **Finish** button.

| Install Wizard : Installation Finished | × |
|---|---|
| Installation completed. When select [Finish], program finishes. | |
| | |
| | |
| | |
| < Bac Finish Cancel | |

Setting up the building tool

The procedure for using the CC78K0R, RA78K0R as the building tool and ID78K0R-QB as the debugging tool is described below.



<1> Select **Project Settings** in the PM+ **Project** menu.

<2> In the **Project Settings** dialog box, click the **Detail Setting** button on the **Tool Version Settings** tab.

| Project Settings | | |
|---|--|--|
| Project Information Source File | | <u>Save</u> Delete |
| Tool CC78K0R RA78K0R SK78K0R ID78K0R-QB | Version W2.12 W1.33 Unused V3.60 | The tool which is not installed with the tool specified by the selected tool set is displayed in the gray. |
| Select only Installed Tools | D <u>e</u> tail Settir | ng |
| | | OK Cancel Help |

<3> In the **Tool Version Detail Setting** dialog box, select the compiler version to use in the "CC78K0R" "RA78K0R" columns and the debugger version to use in the "ID78K0R-QB" column and press "OK" button.

| Tool Version | Detail Setti | ing | | | | |
|----------------------------|---------------------------------|------------------------------------|----------------------------------|-------|--------|--|
| CC78K0R | RA78K0R | SK78K0R | ID78K0R-QB | [| · | |
| Unused ✓₩2.12 □₩2.10 | Unused W1.33 W1.31 | ⊡ V3.10 | Unused ∨3.60 ∪∨3.50 | | | |
| - 78KOR S 78KOR S | oftware Packa oftware Packa | ge V1.10(Englis ge V1.00(Englis | sh Version) sh Version) | | | |
| | | | | | Cancel | |

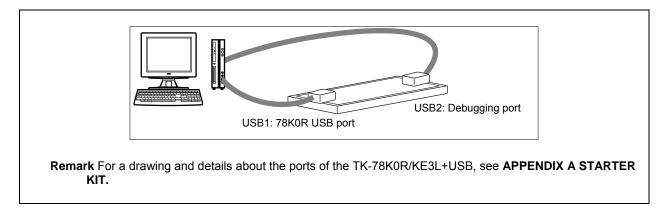
5. 2. 2 Setting up the target environment

Connect the target device to use for debugging.

(1) Connecting the target device

Connect the two USB ports on the TK-78K0R/KE3L+USB to the USB ports of the host by using USB cables.

Figure 5-2 Connecting the TK-78K0R/KE3L+USB



Installing the host driver

The procedure for using the virtual COM port host driver included with the sample driver is described below.

Remark One of the two USB ports on the 78K0R/KE3-L is a debugging port that requires a separate host driver. For details about the files to use and how to obtain them, contact NEC Electronics.

- <1> When the connections of the TK-78K0R/KE3L+USB are recognized by the host, the "Found New Hardware "message is displayed, and then the Found New Hardware Wizard starts.
- <2> [On the first page of the Found New Hardware Wizard dialog box, select No, not this time, and then click the Next button.

| Found New Hardware Wizard | | | |
|---------------------------|--|--|--|
| | Welcome to the Found New Hardware Wizard Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission). Read our privacy policy | | |
| | software? Yes, this time only Yes, now and givery unit is connect a device No, not this time | | |
| | To continue, click Next. | | |
| | < Back Next > Cancel | | |

<3> On the next page, select **Install from a list or specific location (Advanced)** and then click the **Next** button.

| Found New Hardware Wizard | | | | |
|--|---|--|--|--|
| | Welcome to the Found New Hardware Wizard | | | |
| | This wizard helps you install software for: | | | |
| | NEC Electronics KOR Virtual UART | | | |
| If your hardware came with an installation CD or floppy disk, insert it now. | | | | |
| | What do you want the wizard to do? | | | |
| | Install the software automatically (Recommended) Install from a list or specific location (Advanced) | | | |
| | Click Next to continue. | | | |
| | < <u>B</u> ack <u>N</u> ext > Cancel | | | |

<4> On the next page, select **Don't search. I will choose the driver to install** and then click the **Next** button.

| Found New Hardware Wizard |
|---|
| Please choose your search and installation options. |
| Search for the best driver in these locations. |
| Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed. |
| Search removable media (floppy, CD-ROM) |
| ✓ Include this location in the search: |
| Biowse |
| O Don't search. I will choose the driver to install. |
| Choose this option to select the douice anyer from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware. |
| |
| |
| < <u>B</u> ack <u>N</u> ext > Cancel |

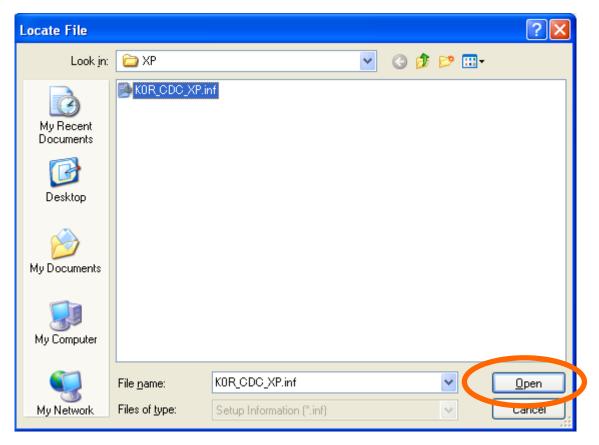
<5> On the next page, click the **Have Disk** button.

| Found New Hardware Wizard |
|--|
| Select the device driver you want to install for this hardware. |
| Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk. |
| Show <u>c</u> ompatible hardware |
| Model |
| NEC Electronics KOR Virtual UART |
| Image: This driver is not digitally signed! Have Disk Tell me why driver signing is important Have Disk |
| < <u>B</u> ack <u>N</u> ext > Jancel |

<6> In the **Install From Disk** dialog box, click the **Browse** button to display the **inf file** folder in the directory in which the sample driver was stored.

| Install F | rom Disk | |
|-----------|---|----------------|
| | Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below. | OK Cancel |
| | Copy manufacturer's files from: | <u>B</u> rowse |

<7> Select the inf file in the **XP** folder according to the OS used on the host, and then click the **Open** button



<8> In the **Install From Disk** dialog box, confirm that the path under **Copy manufacturer's files from:** is correct, and then click the **OK** button.

| Install From Disk | | | X | | |
|-------------------|---|---|--------------|---|--|
| . | Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below. | (| OK Cancel | > | |
| | Copy manufacturer's files from: | | | | |
| | D:\TK-K0R_USB(CDC)\Inf file\XP | * | Browse | | |

<9> In the Found New Hardware Wizard dialog box, select NEC Electronics K0R Virtual UART, and then click the Next button

| Found New Hardware Wizard |
|--|
| Select the device driver you want to install for this hardware. |
| Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk. |
| Show <u>c</u> ompatible hardware |
| Model NEC Electronics K0R Virtual UART |
| |
| This driver is not digitally signed! Have Disk Tell me why driver signing is important Have Disk |
| < <u>B</u> ack <u>Next</u> Cancel |

<10> The driver installation starts.

| Found New | Hardware Wizard | | |
|-----------|--------------------------|-------------------------------|--------|
| Please wa | ait while the wizard ins | stalls the software | |
| Į | NEC Electronics KOR Vi | irtual UART | |
| | | 6 | |
| | | < <u>B</u> ack <u>N</u> ext > | Cancel |

<11> In the Hardware Installation dialog box, click the Continue Anyway button.

| Hardwa | re Installation |
|----------|---|
| <u>.</u> | The software you are installing for this hardware: NEC Electronics KOR Virtual UART has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.) Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing. |
| | Continue Anyway |

<12> The driver is installed. This might take a while depending on the environment.

| Found New Hard | ware Wizard | | | |
|----------------|--------------------------------------|----------------|----------------|--------|
| Please wait wh | ile the wizard installs the | software | | Ø, |
| J NEC | C Electronics K0R Virtual UAR | T | | |
| | usbser.sys To D:\WINDOWS\system32 | \DRIVERS | | |
| | | < <u>B</u> ack | <u>N</u> ext > | Cancel |

<13> On the next page, click the **Finish** button.

| Found New Hardware Wizard | | | | |
|---------------------------|---|--|--|--|
| | Completing the Found New Hardware Wizard The wizard has finished installing the software for: NEC Electronics KOR Virtual UART | | | |
| | < <u>B</u> ack Finish Cancel | | | |

Checking the device assignment

Open the Windows **Device Manager** window. In the **Ports** category, make sure that **NEC Electronics KOR Virtual UART** is displayed and check the assigned COM port number

| 🚇 Device Manager | |
|--|--|
| <u>File Action View H</u> elp | |
| | |
| DHJPC094 DVD/CD-ROM drives IDE ATA/ATAPI controllers Universal Serial Bus controllers Keyboards Computer Sound, video and game controllers System devices Disk drives Monitors Monitors Metwork adapters Muman Interface Devices Processors IDE ATA/ATAPI controllers Processors IDE ATA/ATAPI controllers Processors IDE ATA/ATAPI controllers Ports (COM & LPT) NEC Electronics KOR Virtual UART (COM6) NEC Electronics Starter Kit Virtual UART (COM4) ECP Printer Port (LPT1) Communications Port (COM1) Mice and other pointing devices | |
| | |

Remark Device names and port numbers can be changed. For details, see 6.2 Customizing the Sample Driver.

5.3 On-Chip Debugging

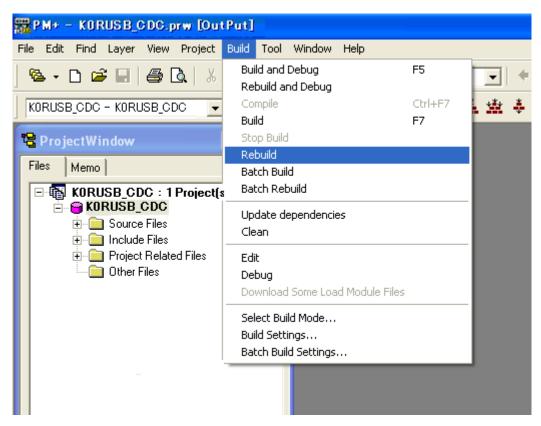
This section describes the procedure for debugging an application program that was developed using the workspace described in **5.2 Setting Up the Environment.**

For the 78K0R/Kx3-L, a program can be written to its internal flash memory and the program operation can be checked by directly executing the program by using a debugger (on-chip debugging).

5. 3. 1 Generating a load module

To write a program to the target device, use a C compiler to generate a load module by converting a file written in C or assembly language.

For PM+, generate a load module by selecting **Rebuild** in the **Build** menu.



5. 3. 2 Loading and executing the load module

Execute the generated load module by writing (loading) it to the target.

(2) Writing the load module

The procedure for writing the load module to the TK-78K0R/KE3L+USB by using PM+ is described below.

<1> Start the ID78K0R-QB by selecting **Debug** in the **Build** menu

| 🧱 PM+ – KORUSB_CDC.prw [Ou | tPut] | |
|--|---|---------|
| File Edit Find Layer View Project | Build Tool Window Help | |
| 🏝 • 🗅 📽 🖬 🚑 🖪 ೫ | Build and Debug Rebuild and Debug | F5 |
| | Compile Build | Ctrl+F7 |
| 염 ProjectWindow | Stop Build | |
| Files Memo | Rebuild Batch Build Batch Rebuild | |
| E-III KORUSB_CDC : 1 Project(: E-IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII | Update dependencies Clean | |
| 🗄 📄 Project Related Files | Edit | |
| Other Files | Debug | |
| | Download Some Load Module Files | |
| | Select Build Mode Build Settings Batch Build Settings | |
| | | |

| Configuration | | X | | |
|--|---------------------|-----------------|--|--|
| Chip <u>Name:</u> uPD 78F1026_64 - | Monitor Clock – | ОК | | |
| | System | Ounce! | | |
| Internal Memory | C User | <u>R</u> estore | | |
| NOM: NOVies | – Fail-safe Break – | <u>P</u> roject | | |
| RA <u>M</u> : 8192* <u></u> Bytes | Detail | Abo <u>u</u> t | | |
| DataFlash: 🔍 💌 KBytes | Detail | <u>H</u> elp | | |
| Main <u>C</u> lock C Clock Socket C External C Sy | stem None | ▼ MHz | | |
| _ <u>S</u> ub Clock (Peripheral) | stem None | ▼ KHz | | |
| Target Device Connection | ID Code | ***** | | |
| Peripheral Break Flash Programming Target | | | | |
| | | Not Connect | | |
| | ₩ide Voltage Flas | n Rewriting | | |
| MI INTERNAL RESET | C Off | | | |
| Memory Mapping | | | | |
| Access Size: | | <u>A</u> dd | | |
| Memory Attribute: Mapping Address: TargetDelete | | | | |
| | | | | |
| | | | | |
| 1 | | | | |

<2> In the **Configuration** dialog box, click on "OK" button.

<3> If a project file included with the sample driver is used, the following dialog box is displayed. Click the **Yes** button to start writing the load module file.

| ID 78KOR-QB | |
|---|--|
| <u>File E</u> dit <u>V</u> iew <u>O</u> ption <u>R</u> ur | i Eve <u>n</u> t Browse Jump <u>W</u> indow <u>H</u> elp |
| $[[[I \models] \models] \models_N] \blacktriangle [\models]) $ | L▲ ≥ ≥ ■ ≈ ≈ ≈ ∞ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ |
| | |
| | |
| | ID78KOR-QB |
| | WF700: Do you want to download Load Module File? |
| | Yes No |

Executing the program

Click the button in the ID78K0R-QB window or select **Run Without Debugging** in the **Run** menu.

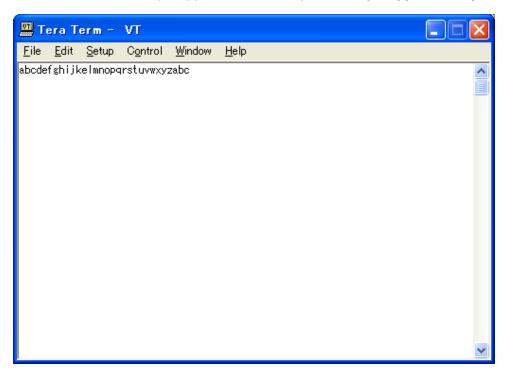
| ID78K0R-QB: K0RUSB_CDC.prj |
|--|
| <u>File Edit View Option Run Event Browse Jump Window Help</u> |
| <u> </u> |
| Source (main.c) |
| Search Watch Quick Refresh Close * 69 void main(void) * * 70 1 INT32 rcv_ret = 0; 71 INT32 snd_ret = 0; * 73 74 cpu_init(); 75 DI(); * 76 DI(); * 77 usbf78k0r_init(); /* initial setting of the USB Function |
| <pre>#</pre> |
| 85 if (dsb1r0k0f_get_bd1rnt(_fret) - bty_thnoh) f 85 if (snd_ret >= 0) { 86 rcv_ret = usbf78k0r_recv_buf(&UserBu 87 } 88 if (rcv_ret >= 0) { 89 snd_ret = usbf78k0r_send_buf(&UserBu 90 } 91 } 82 else { 93 snd ret = 0; |
| |

5.4 Checking the Operation

If the target device that has loaded the sample driver is connected to the host via USB, the result of executing the sample application in the driver can be checked.

Start terminal software (such as Tera Term) on the host, enter the following characters, and then check how they are displayed.

Remark For details of sample application, see Chapter 4 **Sample application specifications.**



CHAPTER 6 USING THE SAMPLE DRIVER

This chapter describes information that you should know when using the USB Communication Device Class sample driver for the 78K0R/Kx3-L.

6.1 Overview

The sample software can be used in the following two ways.

(1) Customizing the sample driver

Rewrite the following sections of the sample driver as required.

. The sample application section in "main.c"

- . The values specified for the various registers in "usbf78k0r.h" file
- . The descriptor information in "usbf78k0r_desc.h" file
- . Device names and provider information included in the virtual COM port host driver (inf file)

Remark: For the list of files included in the sample driver, see **1.1.3 Files included in the sample driver**.

(2) Using functions

Call functions from within the application program as required. For details about the provided functions see **3.3 Function Specifications.**

6. 2 Customizing the Sample Driver

This section describes the sections to rewrite as required when using the sample driver.

6.2.1 Application section

The code in main.c file below includes a simple example of processing using the sample driver. The initialization before and after the processing and endpoint monitoring can be used by including the processing to actually use for the application in this section.

List 6-1 Sample Application Code

```
1
    /*_____
2
     Main function
3
     void main(void)
4
5
     Arguments:
6
         N/A
7
     Return values:
8
         N/A
9
     Overview:
10
         main routine.
11
    */
12
    void main(void)
13
    {
14
         INT32 rcv_ret = 0;
15
         INT32 snd_ret = 0;
16
17
         cpu_init();
18
         DI();
19
20
21
         usbf78k0r_init(); /* initial setting of the USB Function */
22
23
         EI();
24
         while(1)
25
26
         {
              if (usbf78k0r_get_bufinit_flg() != DEV_ERROR) {
27
28
                   if (snd_ret >= 0) {
                       rcv_ret = usbf78k0r_recv_buf(&UserBuf[0], USERBUF_SIZE);
29
30
                   }
                   if (rcv_ret >= 0) {
31
32
                       snd_ret = usbf78k0r_send_buf(&UserBuf[0], rcv_ret);
33
34
              }
35
              else {
                   snd_ret = 0:
36
37
                   rcv_ret = 0;
38
              }
39
         }
40
    }
```

6. 2. 2 Setting up the registers

The registers the sample driver uses (writes to) and the values specified for them are defined in "usbf78k0r.h" file. By rewriting the values in this file according to the actual use for the application, the operation of the target device can be specified by using the sample driver.

6.2.3 Descriptor information

The data the sample driver adds to the USBF during initialization processing (described in **3.1.3 Descriptor settings**) is defined in "usbf78k0r_desc.h" file. Information such as the attributes of the target device can be specified by using the sample driver by rewriting the values in this file according to the use in an actual application.

If the vendor ID and product ID of the device descriptor are rewritten, the vendor ID and product ID must also be rewritten in the host driver to install (the INF file) when connecting the target device. (For details, see **6.2.4 (3) Changing the vendor and product IDs**).

Any information can be specified for the string descriptor. The sample driver defines manufacturer and product information, so rewrite the information as required.

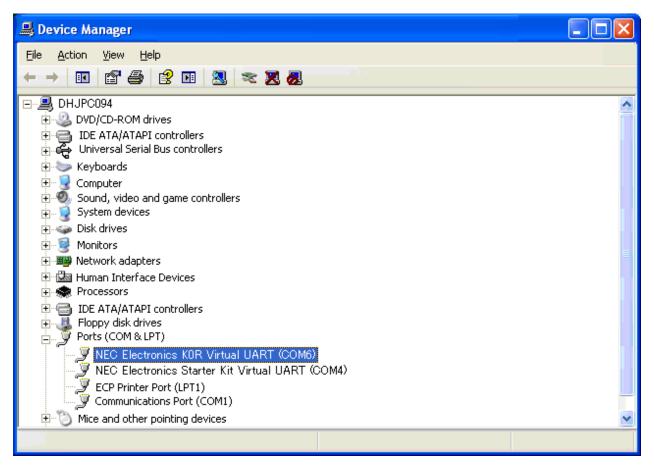
6. 2. 4 Setting up the virtual COM port host driver

The driver that was installed in 5.2.2 Preparing the environment can be customized as follows.

Changing the COM port number

When the connection of a USB device is recognized by the host, the host automatically assigns the COM port number of the device, but the number can be changed to any number. To change the COM port number by using the host, perform the following procedure.

<1> Open the Device Manager Windows and display the "Port" tree in the device list display.



- <2> Select "NEC Electronics Jx3H Virtual UART (COMn)" (where *n* is a number assigned by the host) to display its properties.
- <3> Click the "Advanced" button on the "Port Settings" tab.

| NEC Electronics KOR Virtual UART (COM6) Properties | × |
|--|---|
| General Port Settings Driver Details | |
| | |
| Bits per second: 9600 | |
| Data bits: 8 | |
| Parity: None | |
| Stop bits: 1 | |
| Elow control: None | |
| <u>A</u> dvanced Pustore Defaults | |
| | |
| | |
| | |
| OK Cancel | |

<4> In the "Advanced Settings for COMn" dialog box (where *n* is a number assigned by the host), select any port number from the "COM Port Number" drop-down list.

| Advanced Settings for COM | 16 | | | ? 🗙 |
|---------------------------------|---|---------------|------|------------------|
| | ires 16550 compatible UART) o correct connection problems. or faster performance. | | | OK Cancel |
| <u>R</u> eceive Buffer: Low (1) | | High (14) | (14) | <u>D</u> efaults |
| Transmit Buffer: Low (1) | | High (16) | (16) | |
| COM Port Number: COM6 | ~ | | | |

Remarks 1. Make sure not to select a port number that is used for a different device.

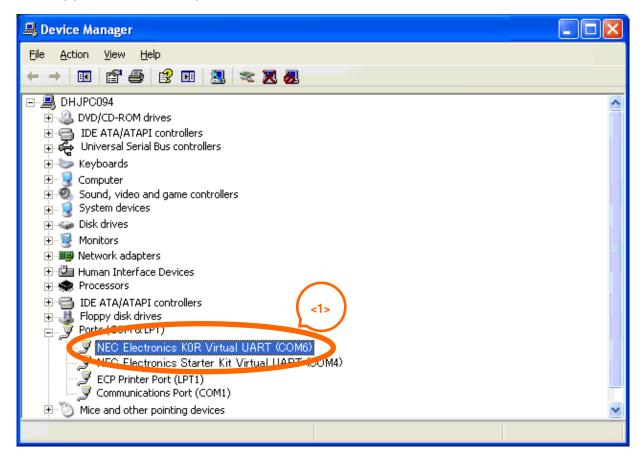
2. Immediately after applying this change, the new port number becomes valid but might not be reflected immediately in the Device Manager.

Changing properties

Some information, such as the attributes of the device used by the Windows Device Manager,

can be changed. The information that can be changed is shown below.

(a) The device name (in the list of devices)



(b) The device name, manufacturer name, and version (in the device properties)

| NEC Electronics KOR Vi | rtual UART (COM 5) Properties | ? 🗙 |
|---|---|--------|
| General Port Settings D | river Details | |
| NEC Electronics KOR Virtual UART (COM6) | | |
| Driver Prov | NEC Electronics Corporation | |
| Driver Date: | 10/15/1999 | |
| Driver Vicen | 5.0.2153.1 | |
| Digital Signer: | Not digitally signed | |
| Driver Details | To view details about the driver files. | |
| Update Driver | To update the driver for this device. | |
| <u>R</u> oll Back Driver | If the device fails after updating the driver back to the previously installed driver. | , roll |
| <u>U</u> ninstall | To uninstall the driver (Advanced). | |
| | ОК | Cancel |

Because this information is displayed based on the information included in the host driver (the INF file), it can be changed by rewriting the INF file. The sections in the INF file, which correspond to the numbers in the example on the previous page, are shown below.

List 6-2INF file "K0R_CDC_XP.inf" code

| 1 | ; .inf file (Win2000,XP): |
|--------|--|
| 2 | [Version] |
| 3 | Signature="\$Windows NT\$" |
| 4 | Class=Ports |
| 5 | ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318} |
| 6 | ClassGuid-{4D50E370-E325-116E-DFC1-06002DE10510} |
| 7 | Provider=%NEC% |
| | |
| 8 9 | LayoutFile=layout.inf |
| | DriverVer=10/15/1999,5.0.2153.1 <3> |
| 10 | |
| 11 | [Manufacturer] |
| 12 | %NEC%=NEC |
| 13 | |
| 14 | |
| 15 | %NEC78K0RKx3L%=Reader, USB¥VID_0409&PID_01D0 |
| 16 | |
| 17 | [Reader_Install.NTx86] |
| 18 | ;Windows2000 |
| 19 | |
| 20 | [DestinationDirs] |
| 21 | DefaultDestDir=12 |
| 22 | Reader.NT.Copy=12 |
| 23 | |
| 24 | [Reader.NT] |
| 25 | CopyFiles=Reader.NT.Copy |
| 26 | AddReg=Reader.NT.AddReg |
| 27 | |
| 28 | [Reader.NT.Copy] |
| 29 | usbser.sys |
| 30 | |
| 31 | [Reader.NT.AddReg] |
| 32 | HKR,,DevLoader,,*ntkern |
| 33 | HKR,,NTMPDriver,,usbser.sys |
| 34 | HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider" |
| 35 | |
| 36 | [Reader.NT.Services] |
| 37 | AddService = usbser, 0x0000002, Service_Inst |
| 38 | |
| 39 | [Service_Inst] |
| 40 | DisplayName = %Serial.SvcDesc% |
| 41 | ServiceType = 1 ; SERVICE_KERNEL_DRIVER |
| 42 | StartType = 3 ; SERVICE_DEMAND_START |
| 43 | ErrorControl = 1 ; SERVICE_ERROR_NORMAL |
| 44 | ServiceBinary = %12%¥usbser.sys |
| 45 | LoadOrderGroup = Base |
| 46 | [Strings] |
| 47 | [Strings] |
| 48 | NEC = "NEC Electronics Corporation" <2> |
| 49 | NEC78K0RKx3L = "NEC Electronics K0R Virtual UART" <1> |
| 50 | Serial.SvcDesc = "USB Serial emulation driver" |
| • | |

Changing the vendor and product IDs

If the vendor and product IDs in the device descriptor are changed, the same changes must be specified in the host driver (the INF file).

Include the vendor and product IDs in the INF file as shown on line 15 in List 6-2.

| Vendor ID: | Represented by four digits in hexadecimal format following "VID_" |
|-------------|---|
| Product ID: | Represented by four digits in hexadecimal format following "PID_" |

6.3 Using Functions

The code for applications can be simplified and the code size can be reduced because frequently used and versatile types of processing are provided as defined functions. For details about each function, see **3.3 Function Specifications.** The following sections of the sample application shown in List can be reused as application examples for various types of defined processing.

(1) Verifying FIFO state for user data

FIFO state notification function (usbf78k0r_get_bufinit_flg) for user data is called and FIFO initialization flag "usbf78k0r_bufinit_flg" for user data is monitored on line 27. This flag is uniquely defined by the sample driver and if FIFO is initialized in the Bus Reset process reported by sample driver INTUSB interrupt and Set Line Coding request process of class request, "1" is set.

"0" is set to clear the error state of transmission/reception process of user data at the FIFO initialization in the sample application.

(2) User data reception processing

For the sample driver, separate functions that define retrieval processing for the received data, one for acquiring the data length and another for copying the data, are provided.

Received data size can be verified before the reception process by calling the acquisition function (usbf78k0r_rdata_length) of reception data length at the reception process based on length of the actually received data. Reception process can also be called on the basis of buffer size when buffer size for user data is determined. However, take care that maximum data length for one time reception should be less than the data size that is received in 1 packet.

In the sample application, data received from used endpoint at the received data in the user data reception function (usbf78k0r_recv_buf) on the line 29 is read as a usage example when buffer size is determined.

(3) User data transmission processing

Used endpoint FIFO state is verified at the transmitted data in the user data transmission function (usbf78k0r_send_buf) on line 32 and if it is FIFO Empty, data is written. In case of FIFO Full, it is error end. When size of the data of the packet transmitted at the earlier and not the transmitted data is Max Packet Size, NULL packet is transmitted. Since this is characteristic of communication device class, NULL packet is transmitted to report that it is last data to host when last packet of data is Max Packet Size.

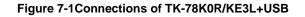
In the sample application, when process is terminated with the generation of error, reception process is stopped and transmission process is repeated until the normal termination of writing of transmission wait data to FIFO. Initialization of FIFO for user data is the only exception. Transmitted/received data and transmission wait data in FIFO are discarded when FIFO is initialized by the request from user or host.

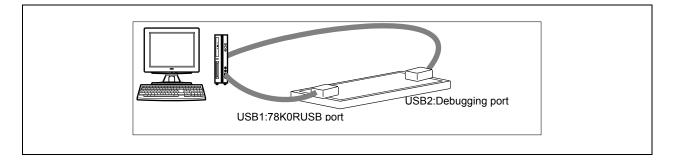
CHAPTER 7 STARTER KIT

This chapter describes the TK-78K0R/KE3L+USB starter kit for the 78K0R/Kx3-L, made by Tessera Technology, Inc.

7.1 Overview

TK-78K0R/KE3L+USB is a kit to develop applications that use the 78K0R/KE3-L. The entire development sequence from creating a program to building, debugging, and checking operation can be performed simply by installing development tools and USB drivers and then connecting either board to the host. This kit uses a monitoring program that enables debugging without connecting an emulator (on-chip debugging).





7.1.1 Features

TK-78K0R/KE3L + USB has the following features.

- A USB miniB connector for the internal USBF
- As small as a business card
- Efficient development by using the board with the integrated development environment (PM+)

7.2 Specifications

The main specifications of the TK-78K0R/KE3L+USB are as follows.

| OCPU | μPD78F1026(78K0R/KE3-L) |
|---------------------|---|
| Operating frequency | 20 MHz (USB:48 MHz) |
| OInterface | USB connector (miniB) x 2 |
| | MINICUBE2 connector |
| | Peripheral board connector x 2 (only the pad) |
| ○Supported platform | Host: DOS/V computer that has a USB interface |
| | OS:Windows XP |
| Operating voltage | 5.0 V(internal operation at 3.3 V) |
| OPackage dimensions | W89 x D52(mm) |

[Memo]

Published by: NEC Electronics Corporation (http://www.necel.com/) Contact: http://www.necel.com/support/