To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

Application Note

V850ES/JG3-H, V850ES/JH3-H
V850ES/JG3-U, V850ES/JH3-U

32-bit Single-Chip Microcontroller

USB CDC (Communication Device Class) Driver

V850ES/JG3-H
- μPD70F3760
- μPD70F3761
- μPD70F3762
- μPD70F3770

V850ES/JH3-H
- μPD70F3765
- μPD70F3766
- μPD70F3767
- μPD70F3771

V850ES/JG3-U
- μPD70F3763
- μPD70F3764

V850ES/JH3-U
- μPD70F3768
- μPD70F3769

Document No. U19599EJ1V0AN00 (1st edition)
Date Published March 2009 N

[MEMO]

PREFACE

Readers

This application note is intended for users who understand the features of the V850ES/Jx3-H or V850ES/Jx3-U, and are going to develop application systems using this product.

Purpose

This application note is intended to give users an understanding of the specifications of the sample driver provided for using the USB function controller incorporated in the V850ES/Jx3-H and V850ES/Jx3-U.

Organization

This application note is broadly divided into the following six sections:

- An overview of the USB function controller incorporated in the V850ES/Jx3-H and V850ES/Jx3-U
- An overview of the USB standard
- The specifications for the sample driver
- The specifications for the sample application
- Development environment
- How to use the sample driver

How to Read This Manual

It is assumed that the readers of this application note have general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

- To learn about the hardware features and electrical specifications of the V850ES/Jx3-H and V850ES/Jx3-U
  → See the separately provided **V850ES/JG3-H, V850ES/JH3-H Hardware User's Manual** and **V850ES/JG3-U, V850ES/JH3-U Hardware User's Manual**.

- To learn about the instructions of the V850ES/Jx3-H and V850ES/Jx3-U
  → See the separately provided **V850ES Architecture User's Manual**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary or decimal ... XXXX
	Hexadecimal ... 0xXXXX

Prefix indicating power 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): $2^{60} = 1,024^6$

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

• Documents Related to the V850ES/Jx3-H and V850ES/Jx3-U

Document Name	Document No.
V850ES Architecture User's Manual	U15943E
V850ES/JG3-H, V850ES/JH3-H Hardware User's Manual	U19181E
V850ES/JG3-U, V850ES/JH3-U Hardware User's Manual	U19287E
V850ES/JG3-H, V850ES/JH3-H, V850ES/JG3-U, V850ES/JH3-U USB CDC (Communication Device Class) Driver Application Note	This manual

• Documents Related to Development Tools (User's Manuals)

Document Name		Document No.
QB-V850ESJX3H In-Circuit Emulator		U19170E
QB-V850MINI On-Chip Debug Emulator		U17638E
QB-MINI2 On-Chip Debug Emulator with Flash Programming Function		U18371E
CA850 Ver. 3.20 C Compiler Package	Operation	U18512E
	C Language	U18513E
	Assembly Language	U18514E
	Link Directives	U18515E
PM+ Ver. 6.30 Project Manager		U18416E
ID850QB Ver. 3.40 Integrated Debugger	Operation	U18604E
SM850 Ver. 2.50 System Simulator	Operation	U16218E
SM850 Ver. 2.00 or Later System Simulator	External Component User Open Interface	U14873E
SM+ System Simulator	Operation	U17199E
	User Open Interface	U17198E
	Basics	U13430E
	Installation	U17419E
	Technical	U13431E
RX850 Ver. 3.20 Real-Time OS	Task Debugger	U17420E
	Basics	U18165E
	In-Structure	U18164E
RX850 Pro Ver. 3.21 Real-Time OS	Task Debugger	U17422E
AZ850 Ver. 3.30 System Performance Analyzer		U17423E
PG-FP5 Flash Memory Programmer		U18865E

CONTENTS

CHAPTER 1 OVERVIEW

This application note describes the CDC (communication device class) sample driver created for the USB function controller incorporated in the V850ES/Jx3-H and V850ES/Jx3-U 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 (USBF) that is incorporated in the V850ES/Jx3-H and V850ES/Jx3-U 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 V850ES/Jx3-H and V850ES/Jx3-U

Endpoint Name	FIFO Size (Bytes)	Transfer Type	Remark
Endpoint 0 Read	64	Control transfer (IN)	–
Endpoint 0 Write	64	Control transfer (OUT)	–
Endpoint 1	64 × 2	Bulk transfer 1 (IN)	Dual-buffer configuration
Endpoint 2	64 × 2	Bulk transfer 1 (OUT)	Dual-buffer configuration
Endpoint 3	64 × 2	Bulk transfer 2 (IN)	Dual-buffer configuration
Endpoint 4	64 × 2	Bulk transfer 2 (OUT)	Dual-buffer configuration
Endpoint 7	8	Interrupt transfer (IN)	–

- 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: 6 MHz external clock multiplied by 8 (48 MHz)

  External clock: Input to the UCLK pin (f_{USB} = 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 CDC sample driver for the V850ES/Jx3-H and V850ES/Jx3-U has the features below. For details about the features and operations, see **CHAPTER 3 SAMPLE DRIVER SPECIFICATIONS**.

- Conforms to the USB CDC 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 3.5 KB
  RAM: About 1.1 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 Driver

Folder	File	Overview
src	`main.c`	Main routine, initialization, sample application
	`usbf850.c`	USB initialization, endpoint control, bulk transfer, control transfer
	`usbf850_communication.c`	CDC-specific processing
include	`errno.h`	Error code definitions
	`main.h`	`main.c` function prototype declarations
	`RegDef.h`	Register definitions
	`Types.h`	User-defined type declarations
	`usbf850.h`	`usbf850.c` function prototype declarations
	`usbf850_communication.h`	`usbf850_communication.c` function prototype declarations
	`usbstrg_desc.h`	Descriptor definitions
	`usbf850_sfr.h`	Macro definitions for accessing the USBF registers
inf file	`JG3H_CDC_VISTA.inf`	INF file for Windows™ Vista™
	`JG3H_CDC_XP.inf`	INF file for Windows XP™

Remark In addition, the project-related files generated when creating a development environment by using the USB-to-serial conversion host driver or 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 the V850ES/Jx3-H and V850ES/Jx3-U

This section describes the V850ES/Jx3-H and V850ES/Jx3-U, which are controlled by using the sample driver.

The V850ES/Jx3-H and V850ES/Jx3-U are products in the low-power series of V850 single-chip microcontrollers for real-time control, made by NEC Electronics. They use a V850 CPU core and have peripherals such as ROM, RAM, timers, counters, a serial interface, an A/D converter, a D/A converter, a DMA controller, a CAN controller, and a USB function controller. For details, see the **V850ES/JG3-H, V850ES/JH3-H Hardware User's Manual** and **V850ES/JG3-U, V850ES/JH3-U Hardware User's Manual**.

1.2.1 Applicable products

The sample driver can be used for the following products:

Table 1-3. V850ES/Jx3-H and V850ES/Jx3-U Products

Generic Name	Part Number	Internal Memory		Incorporated USB Function	Interrupt	
		Flash Memory	RAM[Note 1]		Internal [Note 2]	External [Note 2]
V850ES/JG3-H	μPD70F3760	256 KB	40 KB	Function controller	69	17
	μPD70F3761	384 KB	48 KB	Function controller		
	μPD70F3762	512 KB	56 KB	Function controller		
	μPD70F3770	256 KB	40 KB	Function controller	73	
V850ES/JH3-H	μPD70F3765	256 KB	40 KB	Function controller	69	20
	μPD70F3766	384 KB	48 KB	Function controller		
	μPD70F3767	512 KB	56 KB	Function controller		
	μPD70F3771	256 KB	40 KB	Function controller	73	
V850ES/JG3-U	μPD70F3763	384 KB	48 KB	Function controller Host controller	72	15
	μPD70F3764	512 KB	56 KB	Function controller Host controller		
V850ES/JH3-U	μPD70F3768	384 KB	48 KB	Function controller Host controller		20
	μPD70F3769	512 KB	56 KB	Function controller Host controller		

Notes 1. Includes a data-dedicated 8 KB RAM area.

2. Includes one non-maskable interrupt source.

Caution In this application note, all target microcontrollers are collectively indicated as the V850ES/Jx3-H, unless distinguishing between them is necessary.

1.2.2 Features

The main features of the V850ES/Jx3-H and V850ES/Jx3-U are as follows.

- Memory space: 64 MB of linear address space (for programs and data)
  External expansion: Up to 16 MB (including 1 MB used as internal ROM/RAM space)
- Internal memory: RAM: 40/48/56 KB
  Flash memory: 256/384/512 KB
- External bus interface: Multiplexed bus output (V850ES/JG3-H, V850ES/JG3-U)
  Separate bus/multiplexed bus output selectable (V850ES/JH3-H, V850ES/JH3-U)
  8/16-bit data bus sizing
  Wait function
  - Programmable wait
  - External wait

  Idle state
  Bus hold
- Serial interface: Asynchronous serial interface C (UARTC): 5 shared channels
  Three-wire variable-length serial interface F (CSIF): 2 dedicated channels and 3 shared channels
  I^2C bus interface (I^2C): 3 shared channels
  CAN interface: 1 shared channel (μPD70F3770 and μPD70F3771)
  USB function interface: 1 dedicated channel
  USB host interface: 1 dedicated channel (V850ES/Jx3-U)
- DMA controller: 4 channels
- Clock generator: Main clock or subclock operation:
  Seven-level CPU clock (f_{XX}, $f_{XX}/2$, $f_{XX}/4$, $f_{XX}/8$, $f_{XX}/16$, $f_{XX}/32$, f_{XT})
  Clock-through mode/PLL mode selectable

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 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 (www.usb.org).

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.

Table 2-1. USB 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 periodically 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

2.2 Endpoints

An endpoint is an information unit that is used by the host 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 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 two types of requests: standard requests and class requests.

The sample driver supports the following requests.

(1) Standard requests

Standard requests are used for all USB-compatible devices.

Table 2-2. Standard Requests

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.

(2) 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:

Table 2-3. USB Request Format

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 (the data length)
7		Higher	

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 (Bytes)	Description
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 μA units)

Table 2-6. Interface Descriptor Format

Field	Size (Bytes)	Description
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 CDC sample driver for the V850ES/Jx3-H and the specifications of the functions provided in the V850ES/Jx3-H.

3.1 Overview

3.1.1 Features

The sample driver can perform the following processing:

(1) Initialization

The USBF is set up for use by manipulating various registers. This setup includes specifying settings for the CPU registers of the V850ES/Jx3-H and specifying settings for the registers of the USBF. For details, see **3.2.1 CPU initialization** and **3.2.2 USBF initialization**.

(2) Monitoring endpoints

The statuses of transfer endpoints in the USB function controller are judged by reading the values of various registers. The endpoints for control transfer (endpoint 0) and bulk-out transfer (reception) (endpoint 2) are monitored. During the processing for monitoring endpoint 0, requests are also responded to. For details, see **3.2.3 Monitoring endpoint 0** and **3.2.4 Monitoring endpoint 2**.

(3) Sample application

The data at the endpoint for bulk-out transfer (reception) is read, the data is converted to uppercase or lowercase character strings, 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 V850ES/Jx3-H 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

If a request other than the above is received, the sample driver returns a STALL response.

(2) Class requests

The sample driver responds to class requests of the CDC by using the following 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 `usbf850_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.

Table 3-1. Device Descriptor Settings

Field	Size (Bytes)	Specified Value	Description
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	0x01D0	Product ID: V850ES/JG3-H
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

(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 USBF is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request.

Table 3-2. Configuration Descriptor Settings

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

(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 USBF 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 (Bytes)	Specified Value	Description
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 (Bytes)	Specified Value	Description
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

(4) Endpoint 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 USBF 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.

Table 3-5. Endpoint Descriptor Settings for Endpoint 7

Field	Size (Bytes)	Specified Value	Description
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 3-6. Endpoint Descriptor Settings for Endpoint 2

Field	Size (Bytes)	Specified Value	Description
bLength	1	0x07	Descriptor size: 7 bytes
bDescriptorType	1	0x05	Descriptor type: endpoint
bEndpointAddress	1	0x02	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 1

Field	Size (Bytes)	Specified Value	Description
bLength	1	0x07	Descriptor size: 7 bytes
bDescriptorType	1	0x05	Descriptor type: endpoint
bEndpointAddress	1	0x81	Transfer direction of this endpoint: IN Address of this endpoint: 17
bmAttributes	1	0x02	Transfer type of this endpoint: bulk
wMaxPacketSize	2	0x0008	Maximum packet size of this transfer: 64 bytes
bInterval	1	0x0A	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 USBF.

Table 3-8. String Descriptor Settings

(a) String 0

Field	Size (Bytes)	Specified Value	Description
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 (Bytes)	Specified Value	Description
bLength[Note 1]	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 (Bytes)	Specified Value	Description
bLength[Note 1]	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 (Bytes)	Specified Value	Description
bLength[Note 1]	1	0x16	Descriptor size: 22 bytes
bDescriptorType	1	0x03	Descriptor type: string
bString[Note 2]	20	–	Serial number: 0_98765432

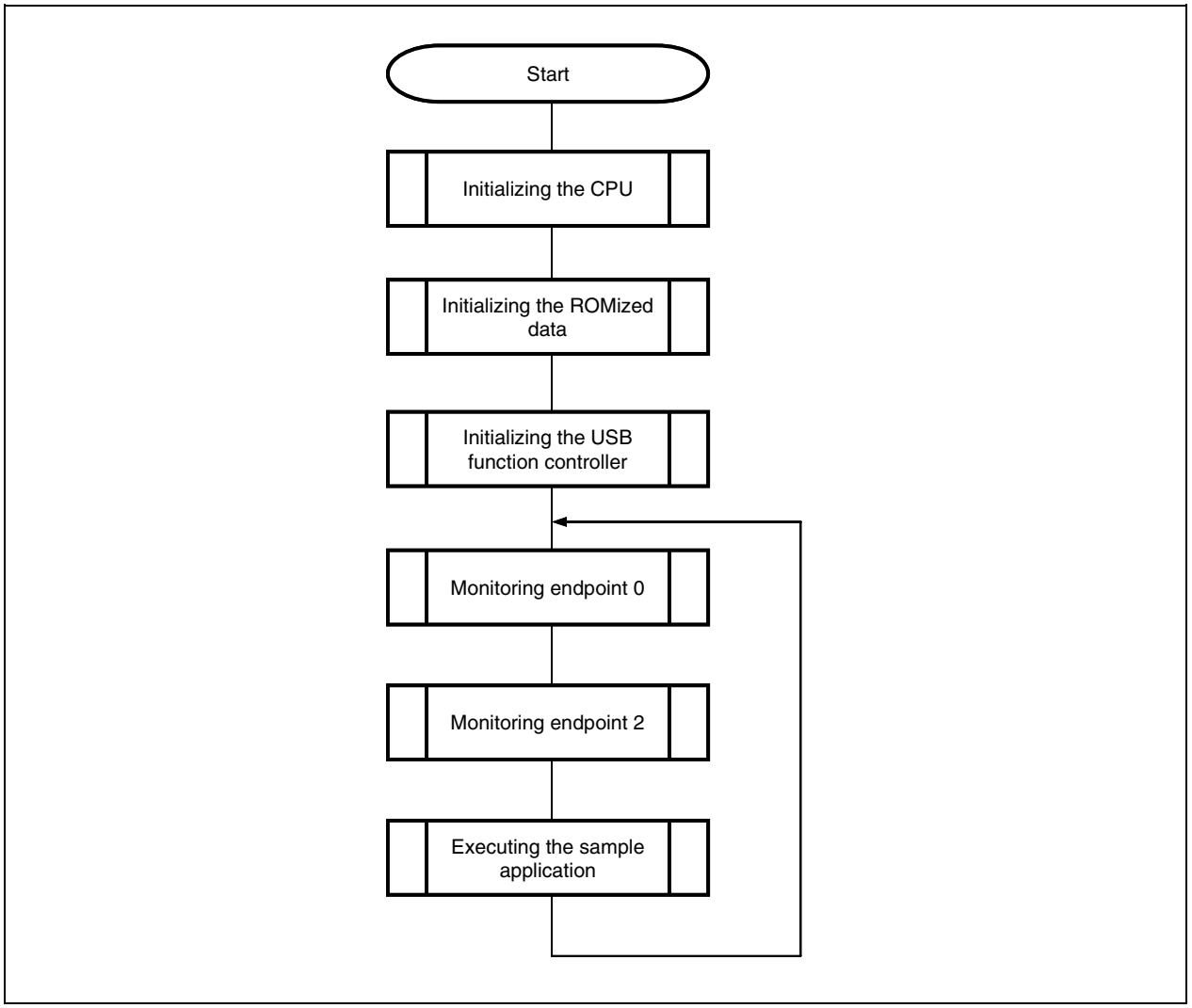
Notes 1. The specified value depends on the size of the bString field.

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

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**.

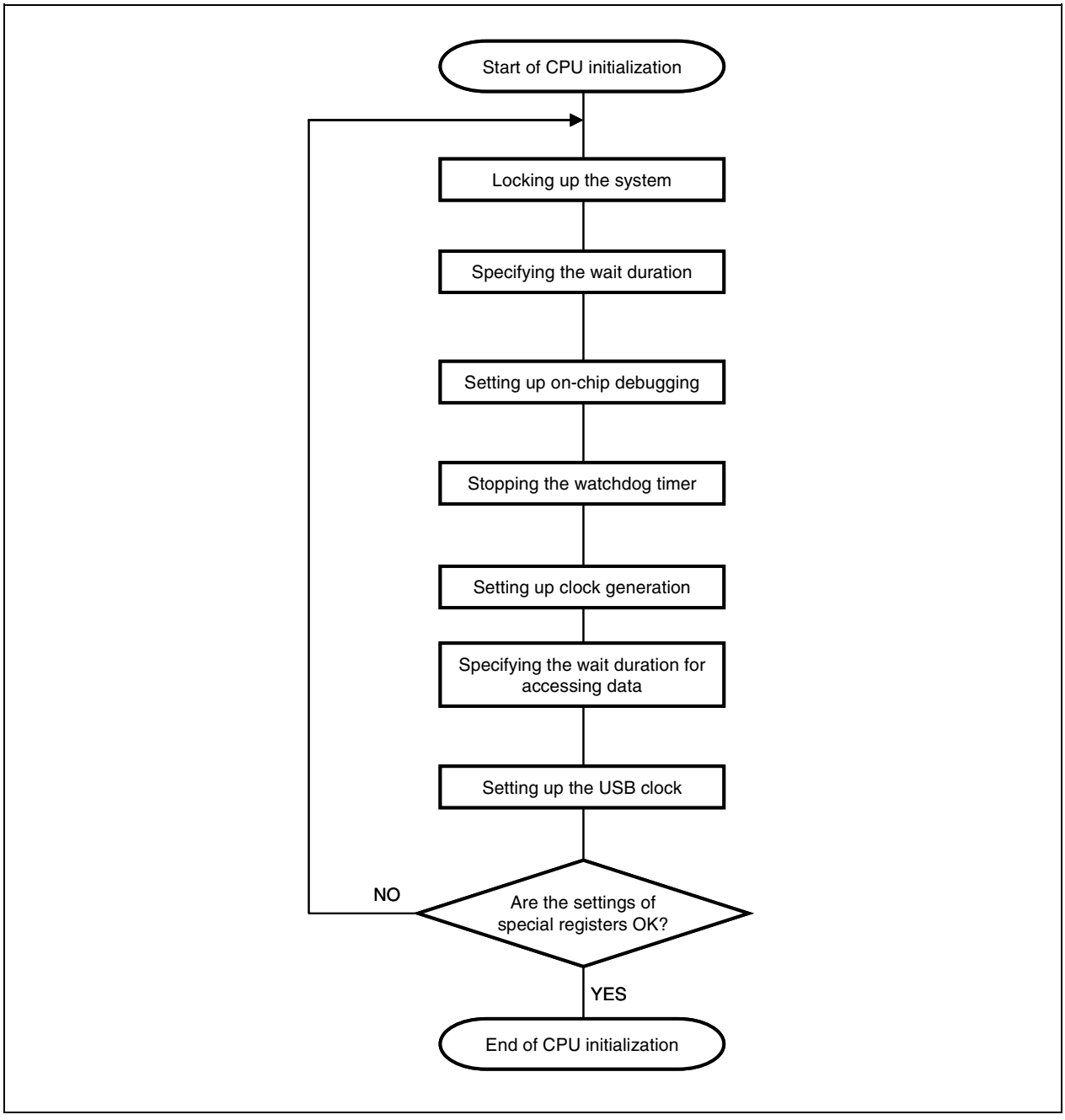
Figure 3-1. Sample Driver Processing Flowchart



3.2.1 CPU Initialization

The settings necessary to use the USBF are specified.

Figure 3-2. CPU Initialization Flowchart



(1) Locking up the system

The system is locked up until the CPU clock frequency stabilizes.

Whether the LOCK bit of the LOCKR register is 0 is monitored.

(2) Specifying the wait duration

The number of clock cycles the system waits for a bus access to the internal peripheral I/O register is specified.

0x12 is written to the VSWC register. For the V850ES/Jx3-H, the number of clock cycles the system waits varies depending on the operation frequency. However, for the sample driver, operation at 33.3 MHz to 48 MHz is assumed.

(3) Setting up on-chip debugging

The CPU operation mode is switched. This setup is necessary for the V850ES/JG3-H and V850ES/JG3-U (but not for the V850ES/JH3-H and V850ES/JH3-U).

1 is written to the OCDM0 bit of the OCDM register to enable the V850ES/JG3-H or V850ES/JG3-U to operate in on-chip debugging mode.

(4) Stopping the watchdog timer

The operation mode of the watchdog timer is switched.

0x00 is written to the WDM21 and WDM20 bits of the WDTM2 register to stop the watchdog timer.

(5) Setting up clock generation

The operation of the internal CPU clock is set up. The following three registers are accessed:

(a) 0x0B is written to the CKC register to multiply the frequency of the clock signal generated by the internal oscillator by 8 by using the PLL.

(b) 0x03 is written to the PLLCTL register to specify PLL mode and start the PLL.

(c) 0x00 is written to the PCC register to specify f_{XX} as the internal clock frequency. For the sample driver, this is specified assuming operation using the main clock.

(6) Specifying the wait duration for accessing data

The number of clock cycles the system waits when I/O ports that operate at different speeds are accessed for data is specified for each I/O port.

0x1171 is written to the DWC0 register to access all three I/O ports while the system waits for one clock cycle.

(7) Setting up the USB clock

The operation of the USBF is set up. The following three registers are accessed:

(a) 0x02 is written to the UCKSEL register to supply the internal clock signal to the USBF.

(b) 0x00 is written to the UFCKMSK register to enable the USBF.

(c) 0x00 is written to the UHCKMSK register to enable the data-dedicated RAM (8 KB) to use for the USBF.

(8) Checking for errors when setting up special registers

Protection errors are checked for.

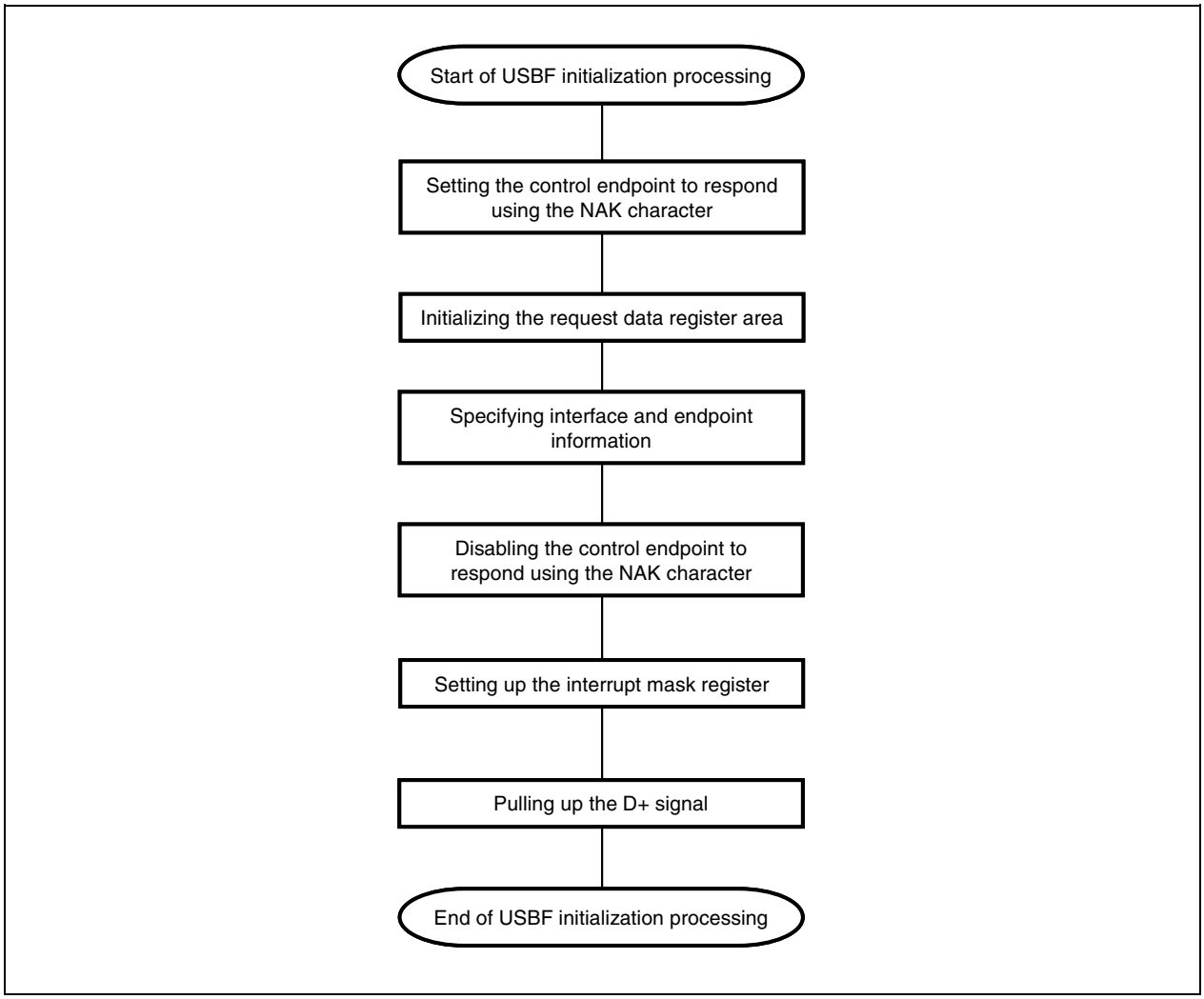
Whether the PRERR bit of the SYS register is 0 is monitored.

A protection error occurs if the OCDM, CKC, or PCC register is manipulated without following the specified procedure.

3.2.2 USBF initialization processing

The settings necessary to use the USBF are specified.

Figure 3-3. USBF Initialization Processing Flowchart



(1) Setting the control endpoint to respond using the NAK character

The NAK response operations for all requests are switched.

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.

The EP0NKA bit is used by software until the data used by requests that are automatically responded to has been added to prevent the hardware from returning unintended data for such requests.

(2) Initializing the request data register area

The descriptor data transmitted in response to a GET_DESCRIPTOR request is added to various registers.

The following registers are accessed:

(a) 0x00 is written to the UF0DSTL register to disable remote wakeup and operate the USBF 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 17).
(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.

(3) Specifying interface and endpoint information

Information such as the number of supported interfaces, whether the alternative setting is used, and the relationship between the interfaces and endpoints is 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 F0AAS 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.

(4) Disabling the control endpoint to respond using the NAK character

The NAK response operations for all requests are switched.
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.

(5) Setting up the interrupt mask registers

Masking is specified for each USBF interrupt source.
The following registers are accessed:

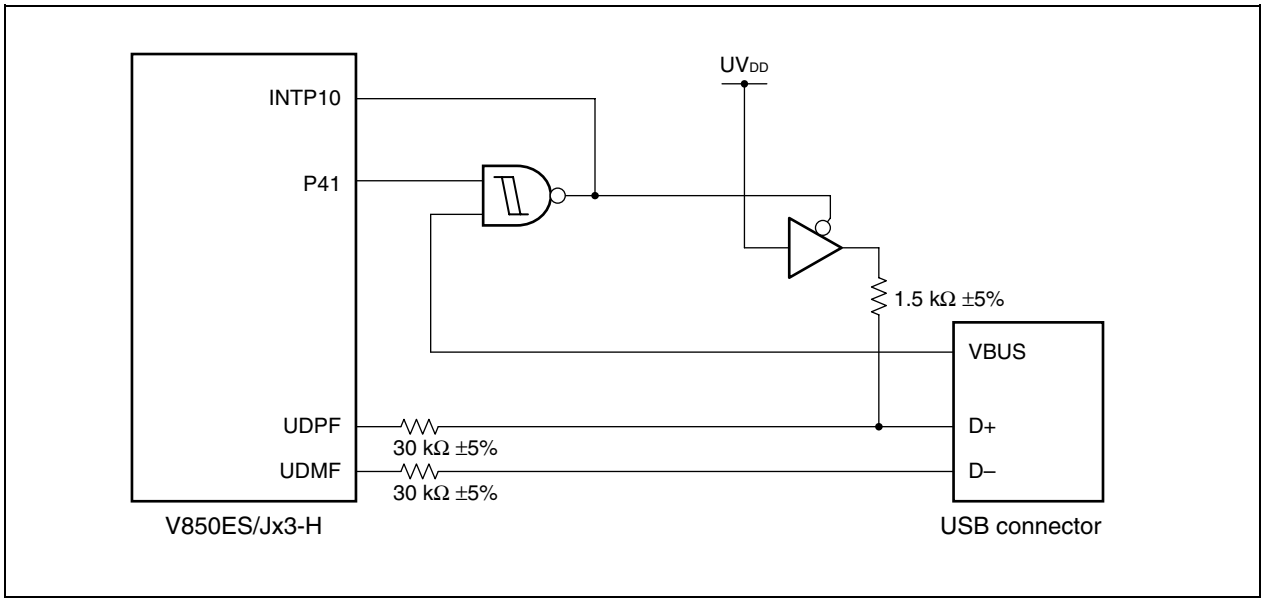
(a) 0x00 is written to the UF0ICn registers (where n = 0 to 7) to clear all interrupt sources.
(b) 0x00 is written to the UF0FICn registers (where n = 0 and 1) to clear all transfer FIFOs.
(c) 0xDF is written to the UF0IM0 register to mask all interrupt sources indicated by the UF0IS0 register.
(d) 0x7E is written to the UF0IM1 register to mask interrupt sources indicated by the UF0IS1 register other than those of the CPUDEC interrupt.
(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) 0x20 is written to the UF0IM4 register to mask all interrupt sources indicated by the UF0IS4 register.

(6) Pulling up the D+ signal

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:

(a) 0xFC is written to the PM4 register of the CPU to set P41 to output mode.
(b) 0x02 is written to the P4 register of the CPU to output 1 from P41.

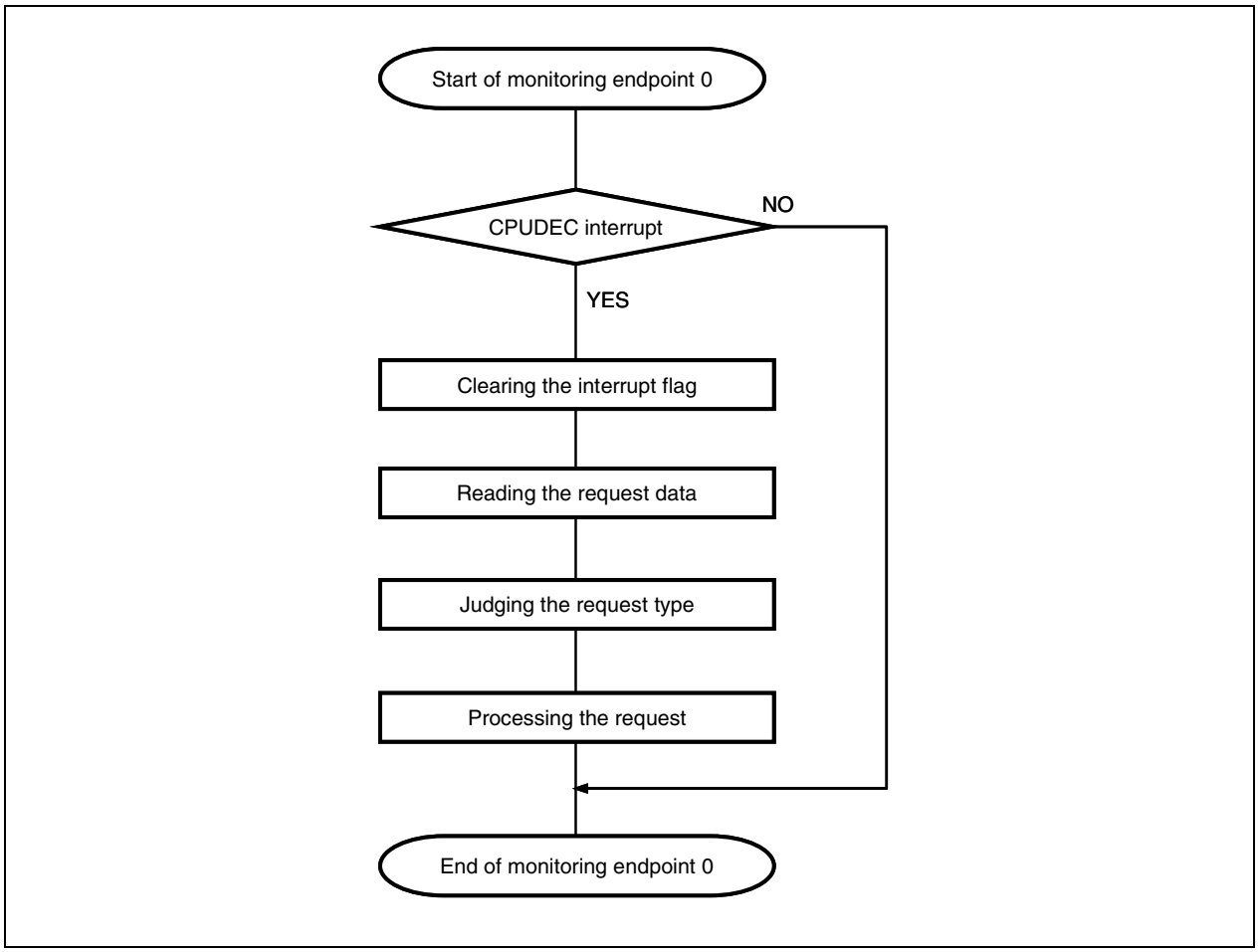
Figure 3-4. USBF Connection Example



3.2.3 Monitoring endpoint 0

Endpoint 0 is used for control transfer. Because some requests, such as standard requests that are used for emulation when a USB device is plugged in, are automatically responded to by hardware, standard requests and class requests to which the hardware does not automatically respond are monitored.

Figure 3-5. Flowchart of Monitoring Endpoint 0



(1) Judging the CPUDEC interrupt

The reception of requests to which the hardware cannot automatically respond is detected.
The following registers are accessed:

(a) Whether the RSUSPD bit of the UF0IS0 register is “1” is monitored. If this bit is set to “1”, it indicates that an interrupt request has been issued.

(b) Whether the CPUDEC bit of the UF0IS1 register is “1” is monitored. If this bit is set to “1”, it indicates that an CPUDEC interrupt request has been issued. This interrupt request is issued if the received request must be decoded. If this interrupt request is issued, it is also reported to the CPU as an INTUSBF0 interrupt.

(2) Clearing the interrupt flag

The flag that indicates that an interrupt request has been issued is cleared.
"0x00" is written to the UF0ICn registers (n = 0 to 4).

(3) Reading request data

The received data is read from the FIFOs and configured as request data.

The request data is acquired as 8-byte data by reading the UF0E0ST register eight times.

(4) Judging the request type

Whether the request to decode is a standard request or class request is judged.

If bits 6 and 5 are “00” for the first byte in the acquired request data (which indicates the request type), they indicate a standard request. If they are “01”, they indicate a class request.

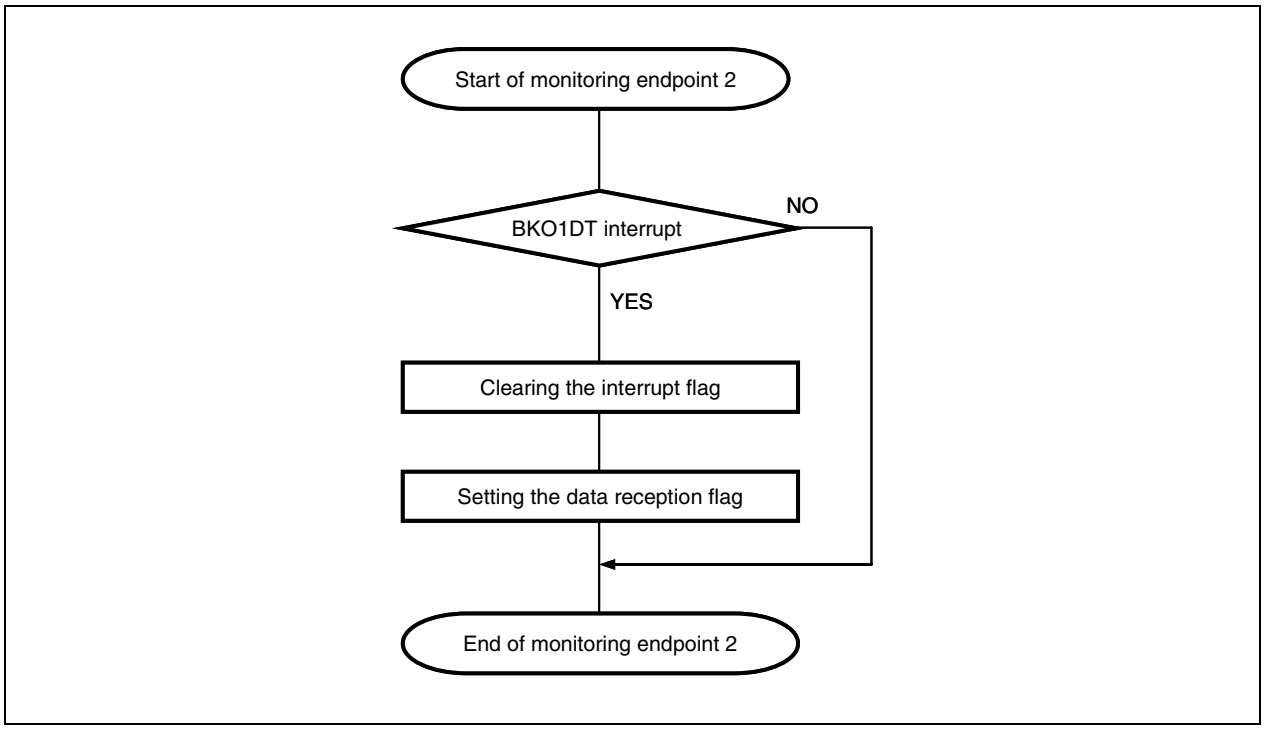
(5) Processing the request

The request is processed according to its type.

3.2.4 Monitoring endpoint 2

Endpoint 2 is used for bulk-out transfer (reception). Whether data has been received is monitored.

Figure 3-6. Flowchart of Monitoring Endpoint 2



(1) Judging the BKO1DT interrupt

Whether data has been successfully received is detected.

The BKO1DT bit of the UF0IS3 register is monitored. If this bit is set to "1”, it indicates that a BKO1DT interrupt request has been issued. This interrupt request is issued if valid data is stored in the reception FIFOs. If this interrupt request is issued, it is also reported to the CPU as an INTUSBF0 interrupt.

(2) Clearing the interrupt flag

The flag that indicates that an interrupt request has been issued is cleared.

"0" is written to the BKO1DTC bit of the UF0IC3 register.

(3) Setting the data reception flag

The flag that indicates the presence of the received data (usbf850_rdata_flg) is set. This flag is uniquely 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	main	Main routine
	init	Initialization routine
	cpu_init	Initializes the CPU.
	romp_init	Initializes ROMization data.
	app_main	Sample application section
usbf850.c	usbf_init	Initializes the USBF.
	intusb0b	Monitors endpoint 0 and controls responses to requests.
	intusb1b	Monitors endpoint 2
	usbf850_data_send	Transmits USB data.
	usbf850_data_receive	Receives USB data.
	usbf850_rdata_length	Acquires the USB received data length.
	usbf850_sendnullEP0	Transmits a NULL packet for endpoint 0.
	usbf850_sendstallEP0	Performs a STALL response for endpoint 0.
	usbf850_standardreq	Processes standard requests.
	usbf850_getdesc	Processes GET_DESCRIPTOR requests.
	usbf850_sstall_ctrl	Controls the STALL responses.
usbf850_communication.c	usbf850_send_encapsulated_command	Processes SendEncapsulatedCommand requests.
	usbf850_set_line_coding	Processes SetLineCoding requests.
	usbf850_get_line_coding	Processes GetLineCoding requests.
	usbf850_set_control_line_state	Processes SetControlLineState requests.
	usbf850_setfunction_communication	Adds CDC class request processing functions.

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 3-7. Calling Functions in the Main Routine

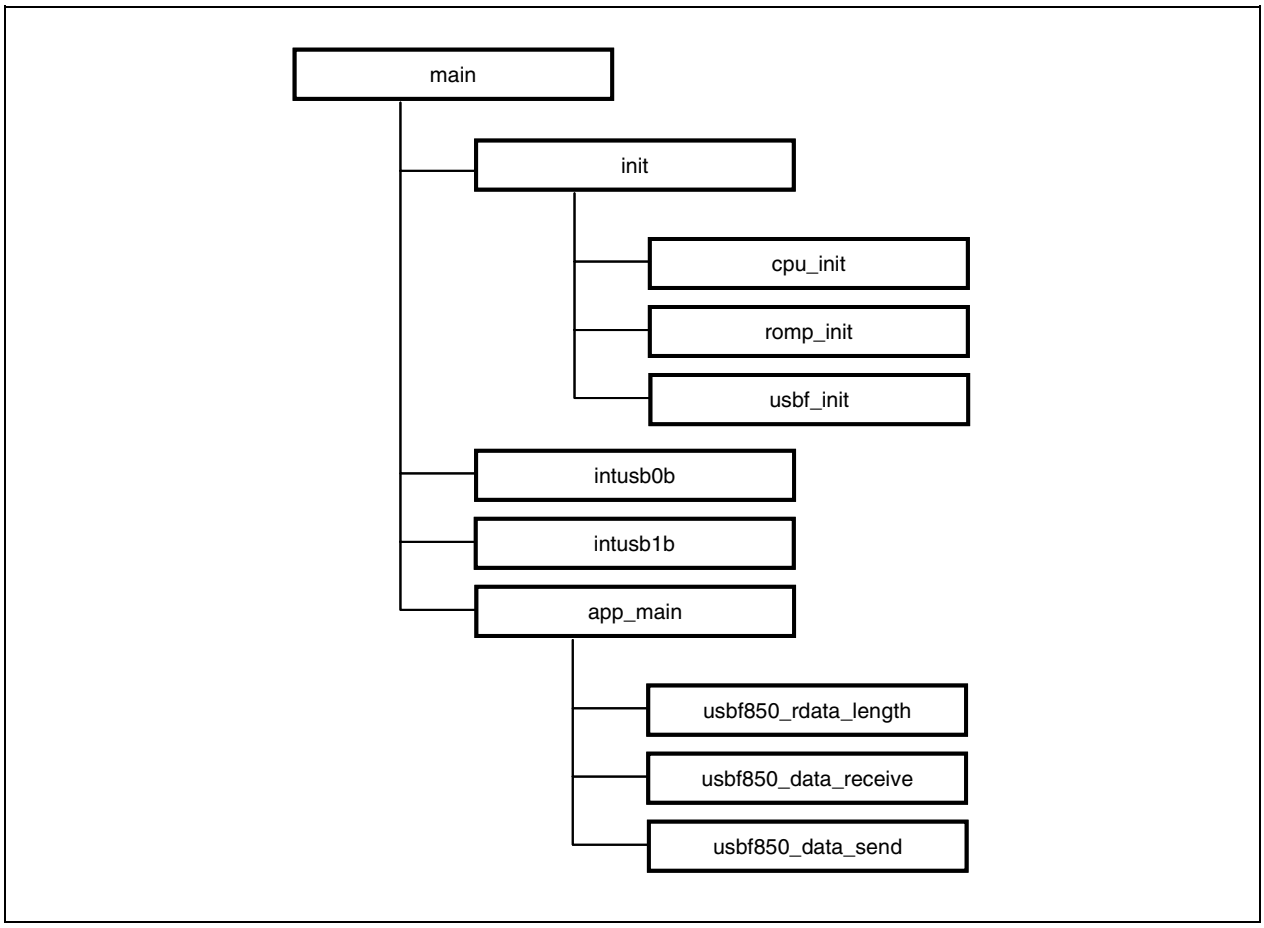


Figure 3-8. Calling Functions During the Processing for the USB Function Controller

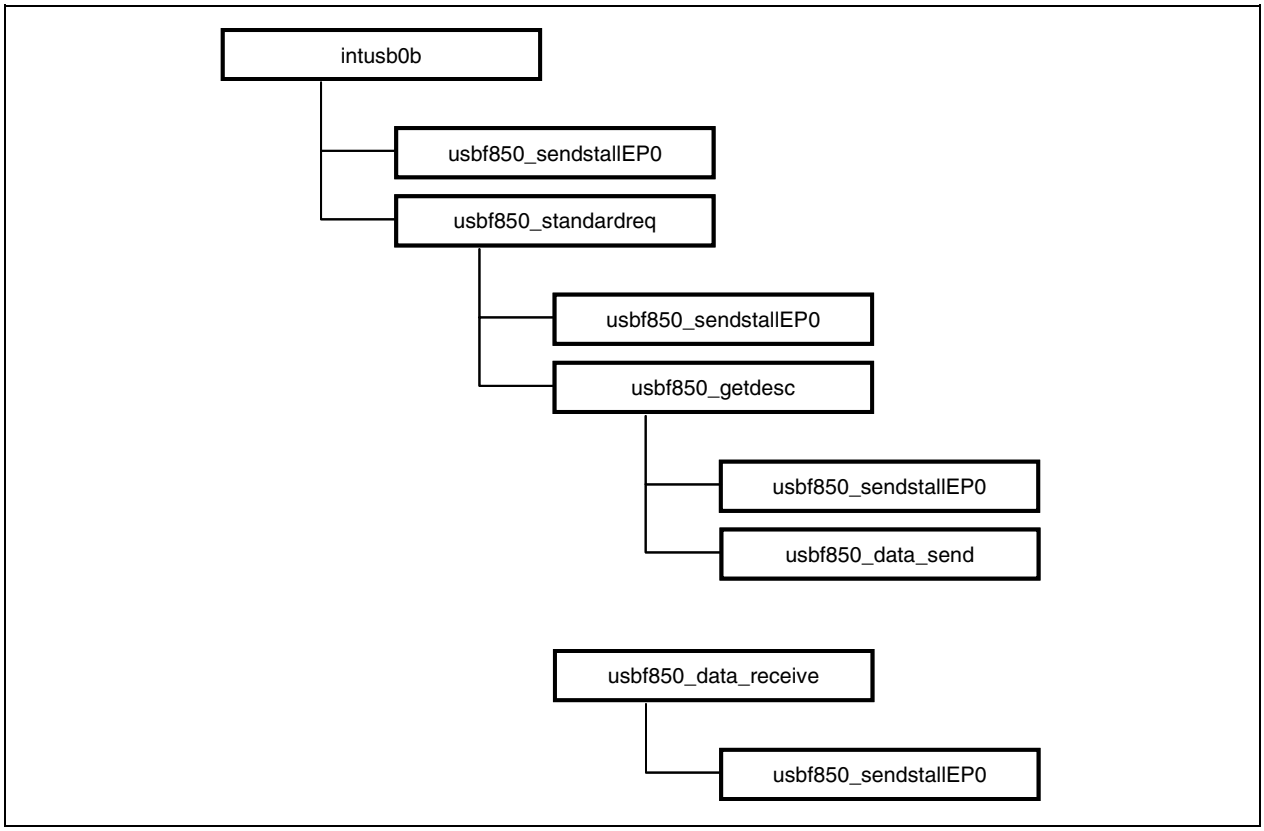
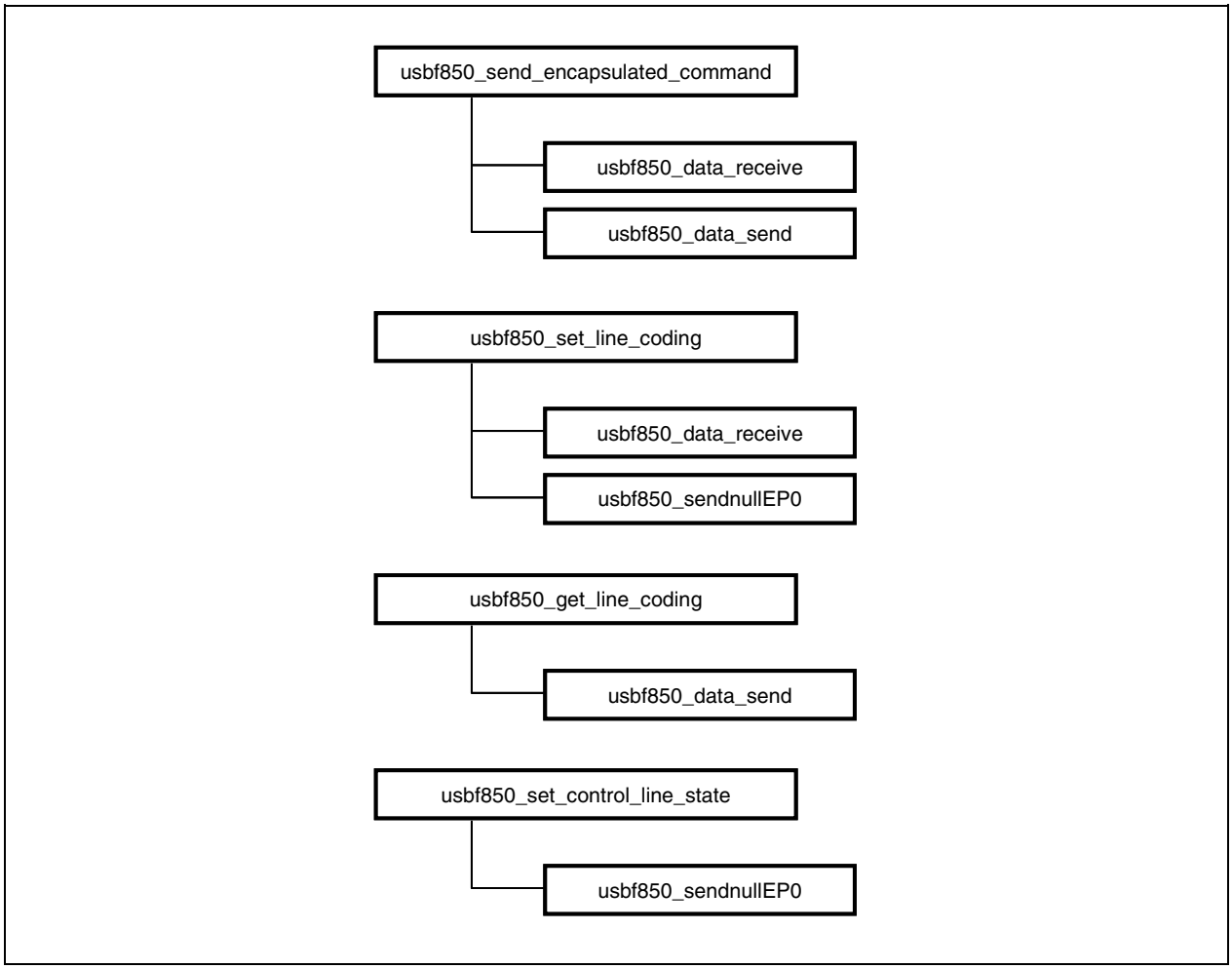


Figure 3-9. Calling Functions During the Processing for the USB Communication Class



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 name	*Parameter summary*

[Return values]

The values returned by the function are described.

Symbol	Description
Return value type and name	*Return value summary*

[Description]

The feature of the function is described.

(2) Functions for the main routine

main

[Overview]

Main processing

[C description format]

```
void main(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called first when the sample driver is executed.

This function calls the initialization function (`init`), the endpoint 0 monitoring function (`intusb0b`), the endpoint 2 monitoring function (`intusb1b`), and then the sample application processing function (`app_main`).

init

[Overview]

Initialization processing

[C description format]

```
void init(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called in the main routine.

The CPU initialization processing function (`cpu_init`), ROMization package initialization processing function (`romp_init`), and then the USBF initialization processing function (`usbf_init`) are called.

cpu_init

[Overview]

Initializes the CPU.

[C description format]

```
void cpu_init(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during initialization.

The settings that are necessary to use the USBF in the V850ES/Jx3-H, such as the number of wait cycles, clock frequency, and operation mode when accessing the bus, are specified.

romp_init

[Overview]

ROMization package initialization processing

[C description format]

```
void romp_init(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during initialization.

The copy function (`_rcopy`) is called and the information stored at the specified address is copied to the RAM area byte by byte.

app_main

[Overview]

Sample application processing

[C description format]

```
void app_main(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during the main processing and is implemented as an example of using the functions included with the sample driver.

After checking whether the USB function controller has received data, the received data length acquisition function (`usbf850_rdata_length`) and the data reception function (`usbf850_data_receive`) are called to retrieve the data. Next, the retrieved data is converted to uppercase or lowercase characters based on the ASCII character codes. Finally, the data transmission function (`usbf850_data_send`) is called to transmit the data.

(3) Functions for the USBF

usbf_init

[Overview]

Initializes the USBF.

[C description format]

```
void usbf_init(void)
```

[Parameters]

None

[Return values]

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.

intusb0b

[Overview]

Monitors endpoint 0.

[C description format]

```
void intusb0b(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during the main processing and supports control transfer.

The issuance of CPUDEC interrupt requests is monitored, and, if the issuance of such a request is identified, request data (8 bytes) is retrieved and then decoded. The request type is judged based on the result of decoding the data, and then the corresponding function is called to issue a response.

intusb1b

[Overview]

Monitors endpoint 2.

[C description format]

```
void intusb1b(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called in the main processing and supports bulk-out transfer (reception).

The issuance of BKO1DT interrupt requests is monitored, and, if the issuance of such a request is identified, the data reception flag (usbf850_rdata_flg) is set.

usbf850_data_send

[Overview]

Transmits USB data.

[C description format]

```
INT32 usbf850_data_send(UINT8 *data, INT32 len, INT8 ep)
```

[Parameters]

Parameter	Description
UINT8 *data	Transmission data buffer pointer
INT32 len	Transmission data length
INT8 ep	Transmission data length

[Return values]

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 endpoint, byte by byte.

usbf850_data_receive

[Overview]

Receives USB data.

[C description format]

```
INT32 usbf850_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 values]

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.

usbf850_rdata_length

[Overview]

Acquires the USB reception data length.

[C description format]

```
void usbf850_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 values]

None

[Description]

This function reads the received data length of the specified endpoint.

usbf850_sendnullEP0

[Overview]

Transmits a NULL packet for endpoint 0.

[C description format]

```
void usbf850_sendnullEP0(void)
```

[Parameters]

None

[Return values]

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.

usbf850_sendstallEP0

[Overview]

Returns a STALL response for endpoint 0.

[C description format]

```
void usbf850_sendstallEP0(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function makes the USBF return a STALL response by setting the bit that indicates the use of STALL handshaking to 1.

usbf850_standardreq

[Overview]

Processes standard requests to which the USBF does not automatically respond.

[C description format]

```
void usbf850_standardreq(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during the processing for monitoring endpoint 0.

If a GET_DESCRIPTOR request is decoded, this function calls the GET_DESCRIPTOR request processing function (`usbf850_getdesc`). For other requests, this function calls the function for returning STALL responses for endpoint 0 (`usbf850_sendstallEP0`).

usbf850_getdesc

[Overview]

Processes GET_DESCRIPTOR requests.

[C description format]

```
void usbf850_getdesc(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function is called during the processing of standard requests to which the USBF does not automatically respond.

If a decoded request requests a string descriptor, this function calls the USB data transmission function (`usbf850_data_send`) 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 for endpoint 0 (`usbf850_sendstallEP0`).

usbf850_sstall_ctrl

[Overview]

Controls STALL responses.

[C description format]

```
void usbf850_sstall_ctrl(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function calls the STALL response processing function for endpoint 0 (`usbf850_sendstallEP0`).

(4) Functions for USB communication device class processing

usbf850_send_encapsulated_command

[Overview]

Processes SendEncapsulatedCommand requests.

[C description format]

```
void usbf850_send_encapsulated_command(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function calls the data reception function (`usbf850_data_receive`) to retrieve the data received at endpoint 0, and then calls the data transmission function (`usbf850_data_send`) to transmit data from endpoint 2 via bulk-in transfer (transmission).

usbf850_set_line_coding

[Overview]

Processes SetLineCoding requests.

[C description format]

```
void usbf850_set_line_coding(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function calls the data reception function (`usbf850_data_receive`) to retrieve the data received at endpoint 0, and then writes the data to the UART_MODE_INFO structure. Based on the retrieved data, this function specifies the transfer speed, data length, and UART mode, and then calls the NULL packet transmission function for endpoint 0 (`usbf850_sendnullEP0`).

usbf850_get_line_coding

[Overview]

Processes GetLineCoding requests.

[C description format]

```
void usbf850_get_line_coding(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function calls the data transmission function (`usbf850_data_send`) to transmit the value of the UART_MODE_INFO structure from endpoint 0.

usbf850_set_control_line_state

[Overview]

Processes SetControlLineState requests.

[C description format]

```
void usbf850_set_control_line_state(void)
```

[Parameters]

None

[Return values]

None

[Description]

This function calls the NULL packet transmission function for endpoint 0 (`usbf850_sendnullEP0`).

usbf850_setfunction_communication

[Overview]

Adds CDC class request functions.

[C description format]

```
void usbf850_setfunction_communication(void)
```

[Parameters]

None

[Return values]

None.

[Description]

This function adds the addresses of various functions for processing USB communication class requests.

CHAPTER 4 SAMPLE APPLICATION SPECIFICATIONS

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

4.1 Overview

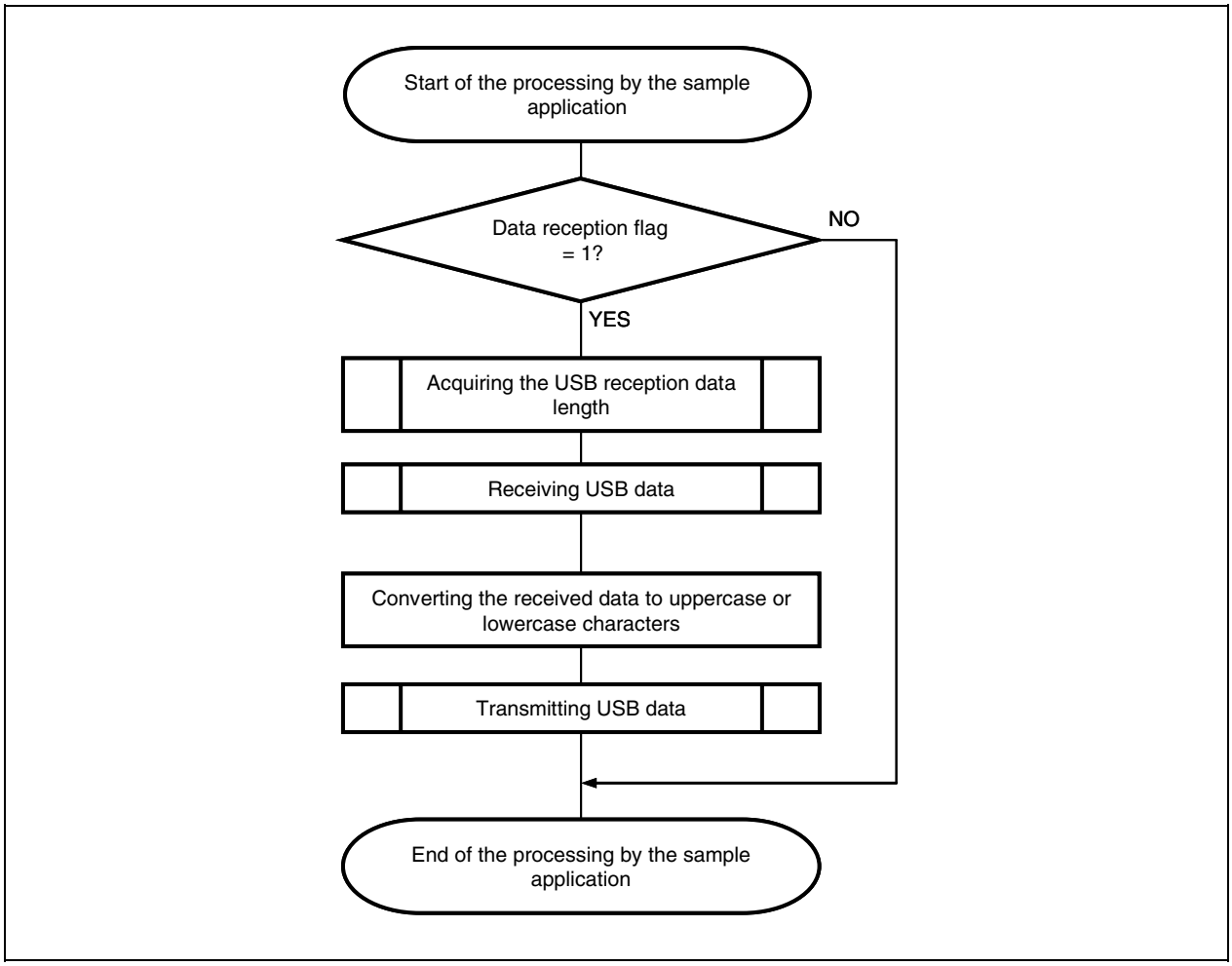
The sample application is provided as a simple example of using the USB communication device class driver and is incorporated in the main routine of the sample driver.

The sample application reads the data received by the USB function controller, converts uppercase characters to lowercase characters based on the ASCII character codes, and then transmits the converted 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) Checking for received USB data

The data reception flag (usbf850_rdata_flg) set by the sample driver is monitored. If this flag is set, it indicates that there is received data in the USB function controller.

(2) Acquiring the USB reception data length

The reception data length is acquired. A sample driver function is called for this processing.

(3) Receiving USB data

Reception starts by specifying the buffer (FIFO) in which to store the received data, the length of the data to receive, and the reception endpoint number. A sample driver function is called for this processing.

(4) Converting the received data to uppercase or lowercase characters

Data is read from the reception FIFOs and characters are converted. The letters “A” to “Z” and “a” to “z” are identified based on their ASCII character codes, and then uppercase characters are converted to lowercase characters and vice versa. 2-byte characters, numbers, and symbols are not converted.

(5) Transmitting USB data

The converted data is transmitted. Transmission starts by specifying the buffer in which the data to transmit is stored, the length of the data, and the transmission endpoint number. A sample driver function is called for this processing.

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**.

List 4-1. Sample Application Code Portion

```
1       #include "main.h"
2       #include "usbf850.h"
3       #include "usbf850_sfr.h"
4       #include "RegDef.h"
5       #include "errno.h"
6                :
7               Omitted
8                :
9       void app_main( void )
10      {
11                  UINT8 user_data[ USER_RBUF_SIZE ];
12                  INT32 ret;
13                  INT32 len;
14                  INT32 i;
15                       :
16                      Omitted
17                       :
18                  memset( user_data , 0 , USER_RBUF_SIZE );
19                  usbf850_rdata_length( &len , C_BKO1 );
20                  ret = usbf850_data_receive(&user_data[ 0 ], len, C_BKO1);
21                  if ( ret != DEV_ERROR ) {
22                       :
23                      Omitted
24                       :
25                  usbf850_data_send(user_data, len, C_BKI1);
```

(1) Definitions and declarations

`usbf850.h` and `usbf850_sfr.h` are included in order to use the sample driver functions.

(2) Acquiring the reception data length and receiving data

The function for acquiring the data length (`usbf850_rdata_length`) is called on line 19 and the function for receiving data (`usbf850_data_receive`) is called on line 20. C_BKO1, which is used as an argument indicating the endpoint number, is defined in `usbf850_sfr.h`.

(3) Transmitting data

The function for transmitting data (`usbf850_data_send`) is called on line 25. C_BKI1, which is used as an argument indicating the endpoint number, is defined in `usbf850_sfr.h`.

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 V850ES/Jx3-H and the procedure for debugging the application.

5.1 Used Products

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:

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

Components		Product Example	Remark
Hardware	Host	–	A PC/AT™-compatible computer using Windows XP or Windows Vista
Software	Integrated development tool	PM+	V6.31
	Compiler	CA850	W3.20

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	–	A PC/AT-compatible computer using Windows XP or Windows Vista
	Target device	TK-850/JG3H	Tessera Technology, Inc.
	USB cables	–	miniB-to-A connector cable
Software	Integrated development tool	PM+	V6.31
	Debugger	ID850QB	V3.50
	Flash memory programming tool	ID850QB	V3.50
Files	Device file	DF703771	For the V850ES/Jx3-H
	Host driver for the debugging port	–	**Note 1**
	Project files	–	**Note 2**

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 Used Products**.

5.2.1 Preparing the host environment

Create a dedicated workspace on the host for debugging.

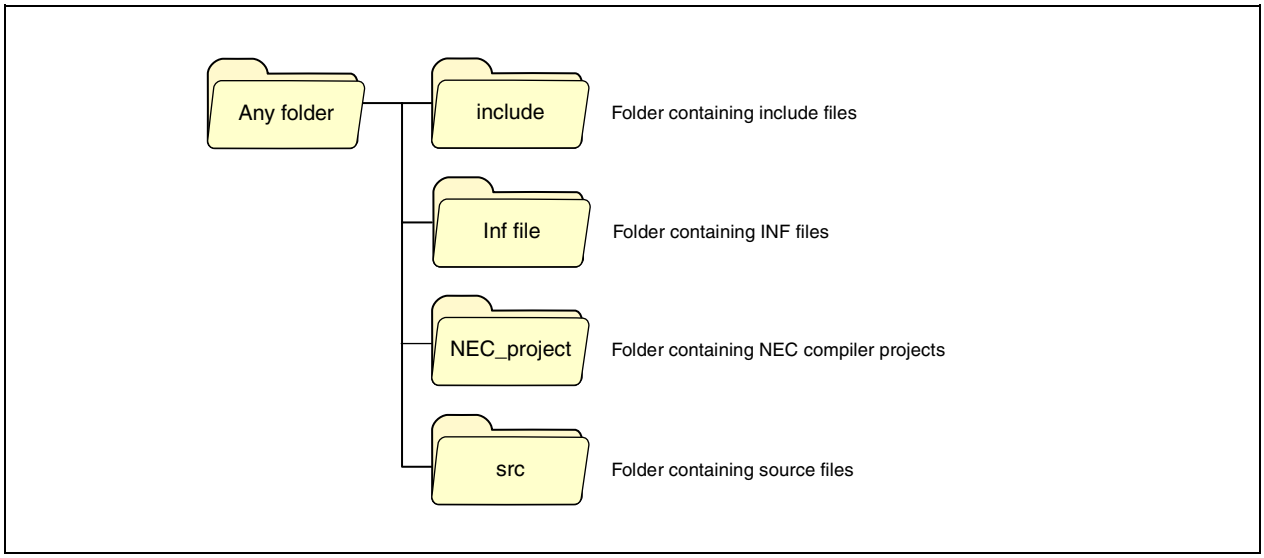
(1) Installing an integrated development tool

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

(2) 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..

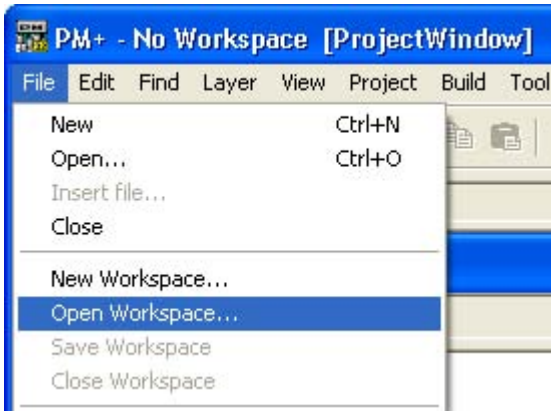
Figure 5-1. Folder Structure of the Sample Driver



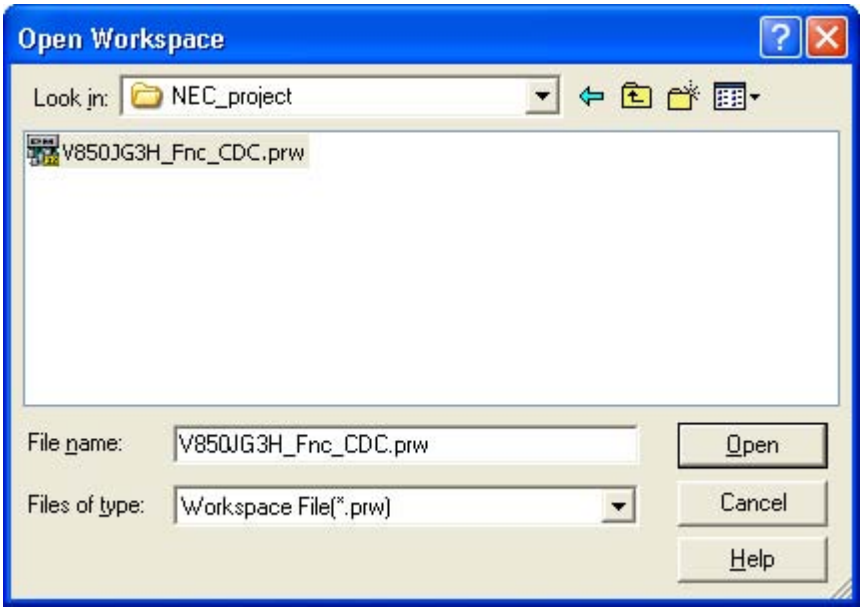
(3) Setting up the workspace

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.



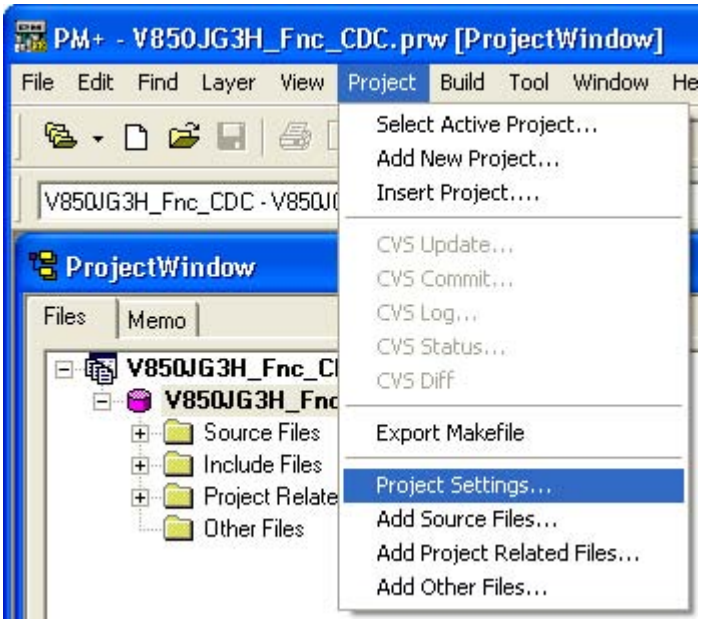
<2> In the **Open Workspace** dialog box, specify the workspace file in the NEC_project folder, which is the sample driver installation directory.



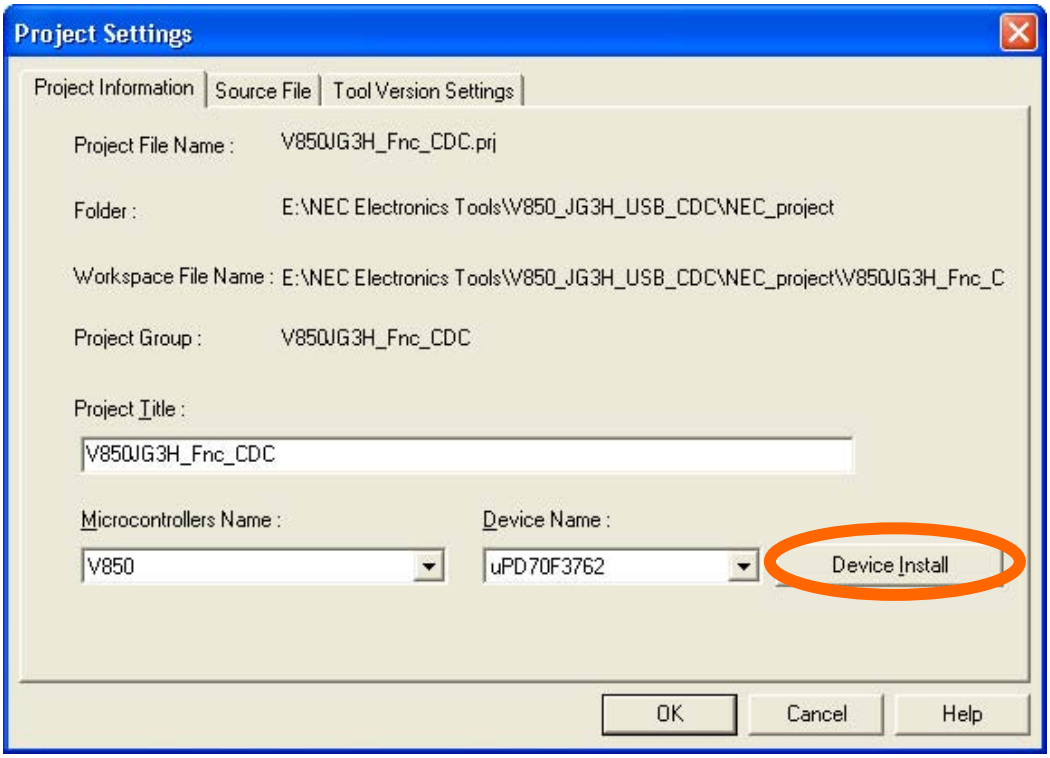
(4) Installing a device file

The procedure for using a device file for the V850ES/Jx3-H is described below.

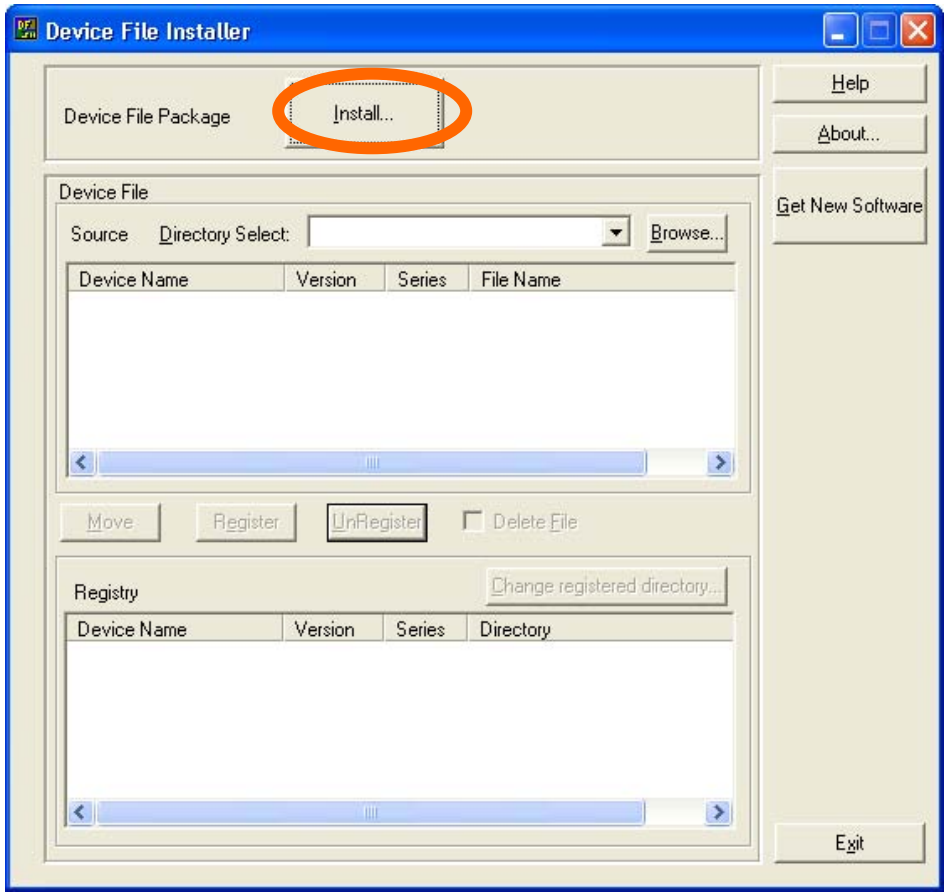
<1> Select **Project Settings** in the PM+ **Project** menu.



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

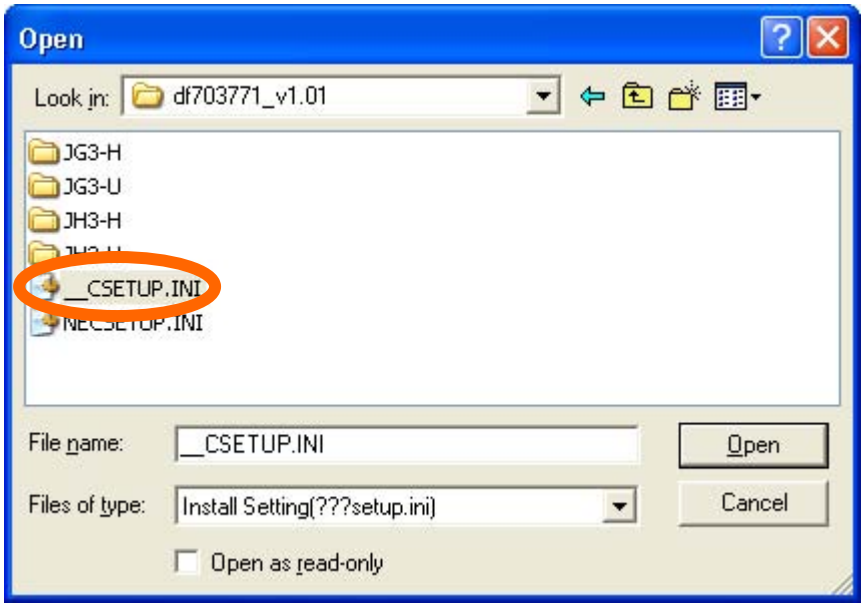


<3> In the **Device File Installer** dialog box, click the **Install** button to start the installation wizard.



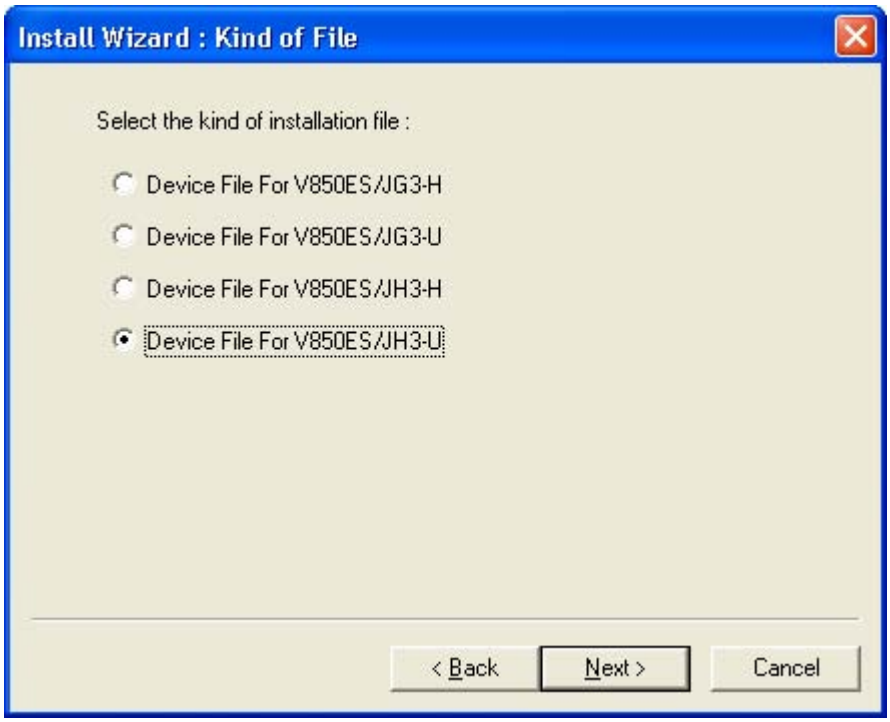
<4> In the **Install Information File** dialog box, click the **Browse** button.

<5> In the **Open** dialog box, open the directory in which the device file was stored, select *__CSETUP.INI*, and then click the **Open** button.

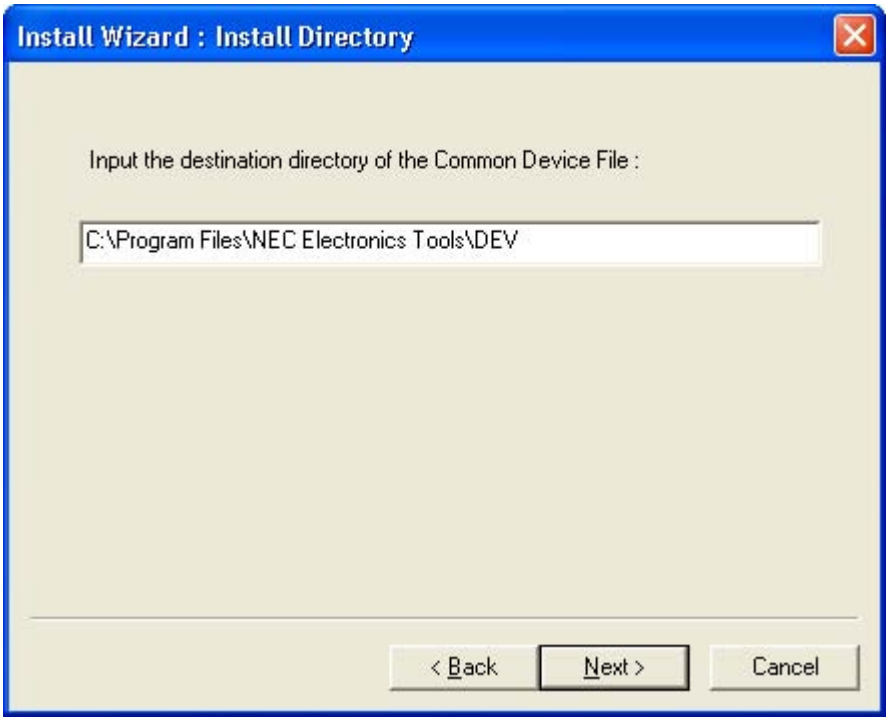


<6> In the **Install Information File** dialog box, click the **Next** button.
In the **NEC SOFTWARE LICENSE AGREEMENT** dialog box, read the license agreement, and then click the **Agree** button if you agree with the terms.

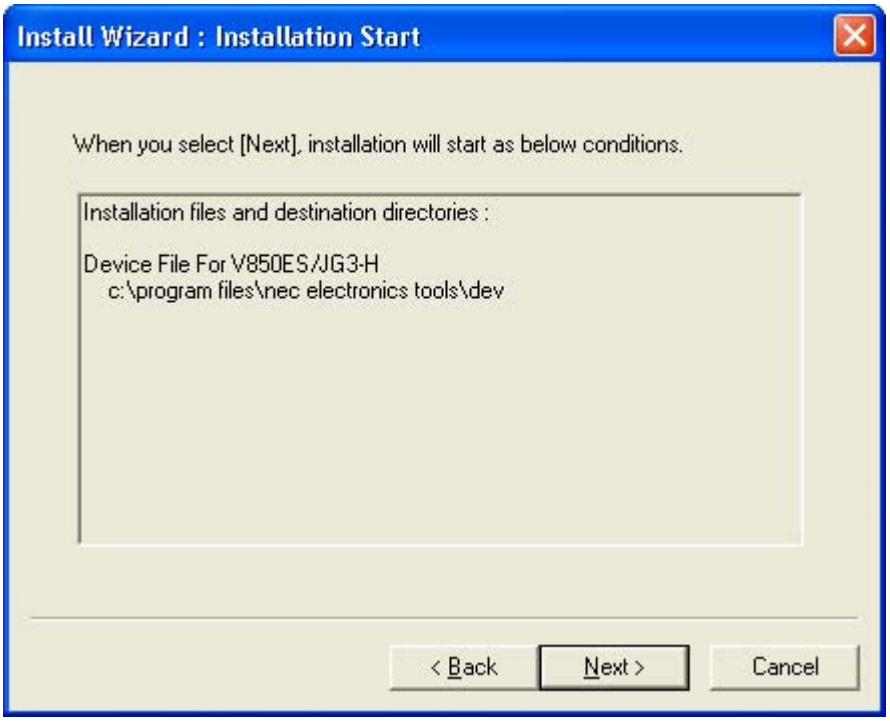
<7> In the **Kind of File** dialog box, select the device file to install, and then click the **Next** button.



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

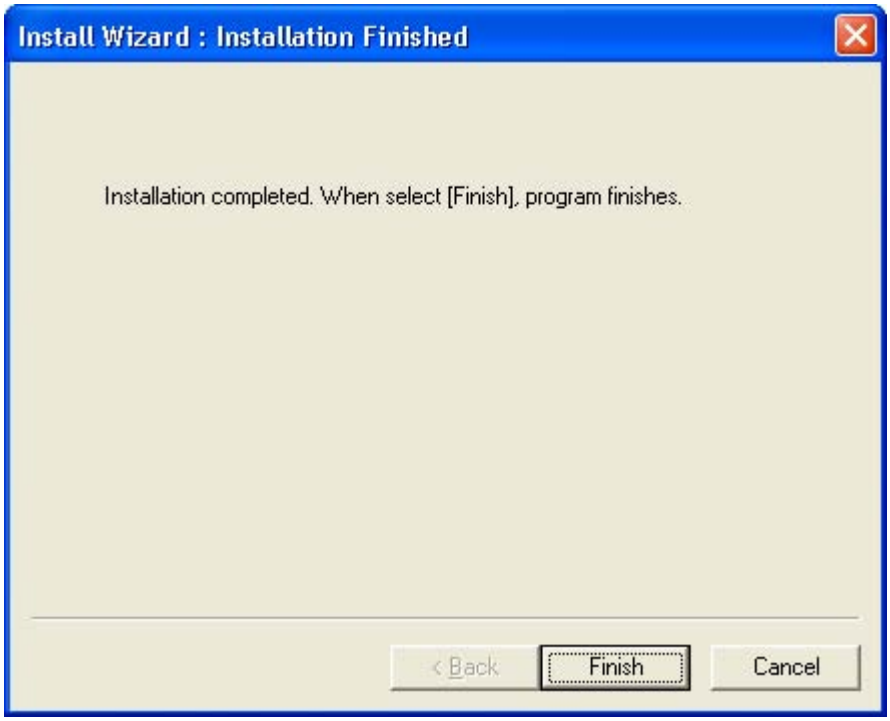


<9> In the **Installation Start** dialog box, click the **Next** button.



<10> The device file is installed to the project. This might take a while depending on the environment.

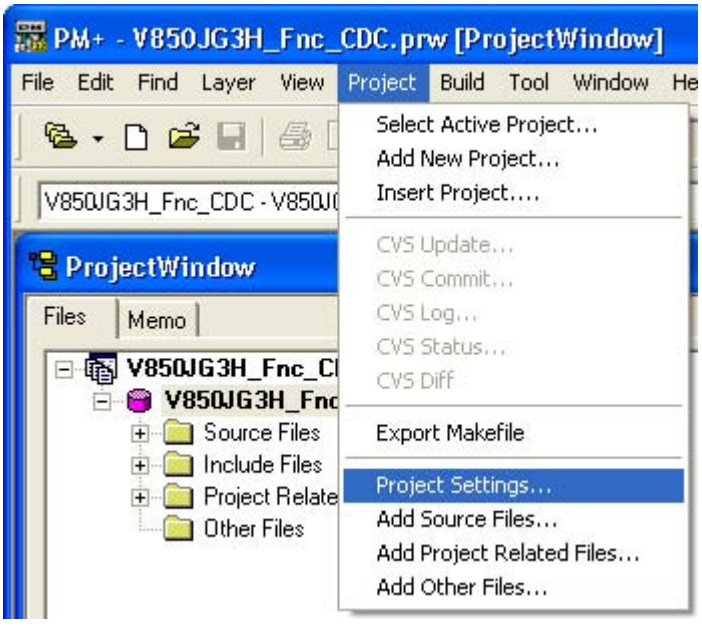
<11> In the **Installation Finished** dialog box, click the **Finish** button.



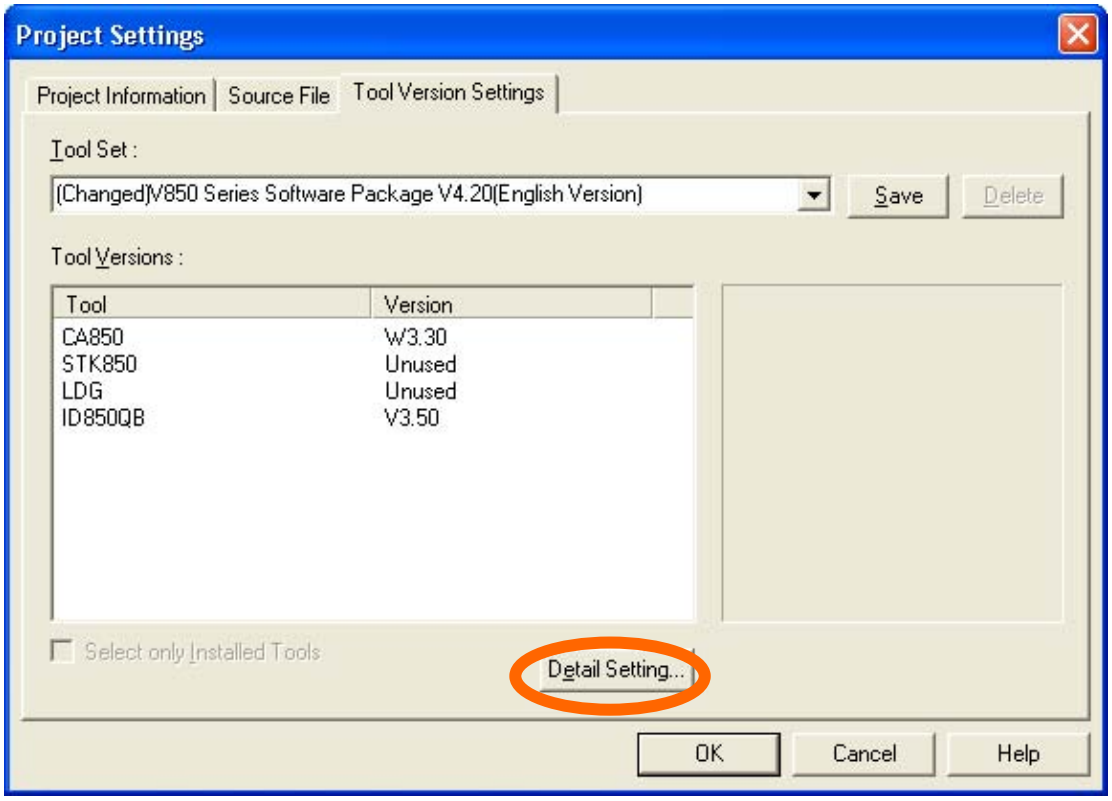
(5) Setting up the building tool

The procedure for using the CA850 as the building tool and the ID850QB as the debugging tool is described below.

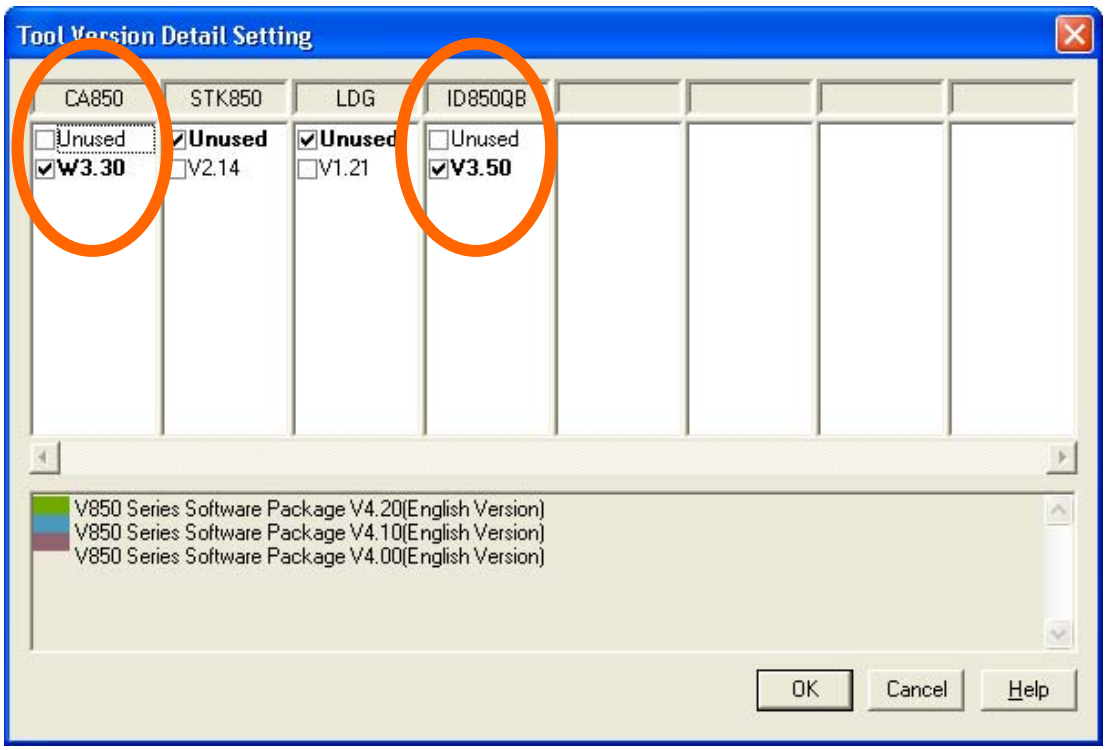
<1> Select **Project Settings** in the PM+ **Project** menu.



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



<3> In the **Tool Version Detail Setting** dialog box, select the compiler version to use in the **CA850** column and the debugger version to use in the **ID850QB** column.



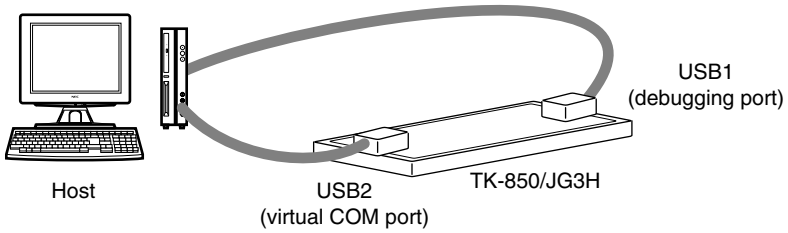
5.2.2 Setting up the target environment

Connect the target device to use for debugging.

(1) Connecting the target device

Connect the two USB ports on the TK-850/JG3H to the USB ports of the host by using USB cables.

Figure 5-2. Connecting the TK-850/JG3H



Remark For a drawing and details about the ports of the K-850/JG3H, see **APPENDIX A STARTER KIT**.

(2) Installing the host driver

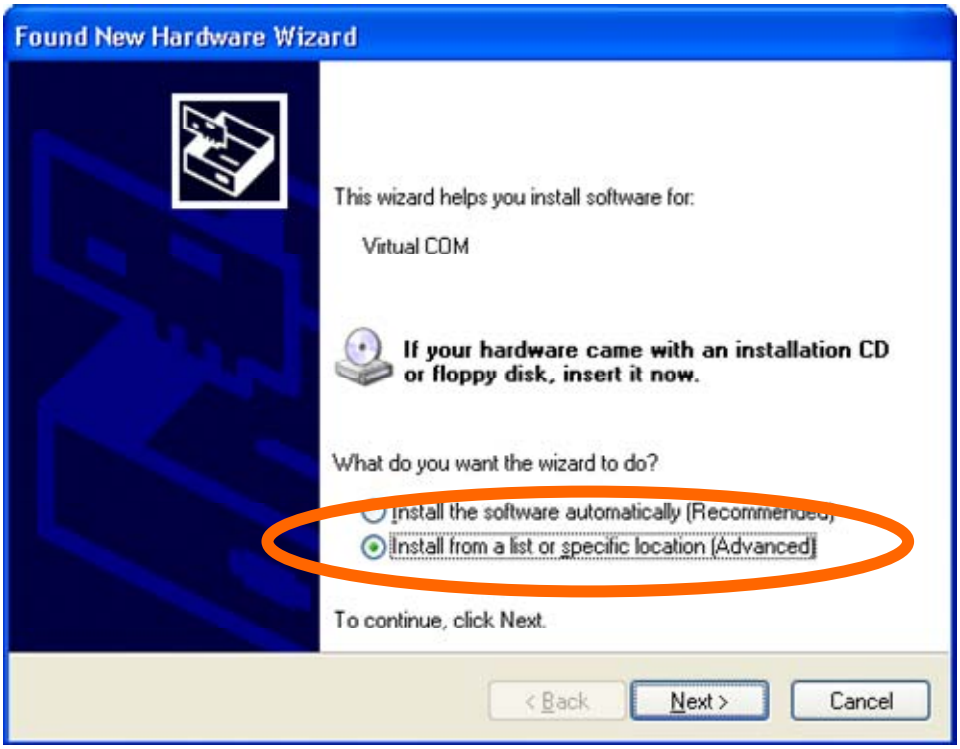
The procedure for using the virtual COM port host driver included with the sample driver is described below.

Remark One of the two USB ports on the TK-850/JG3H 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.

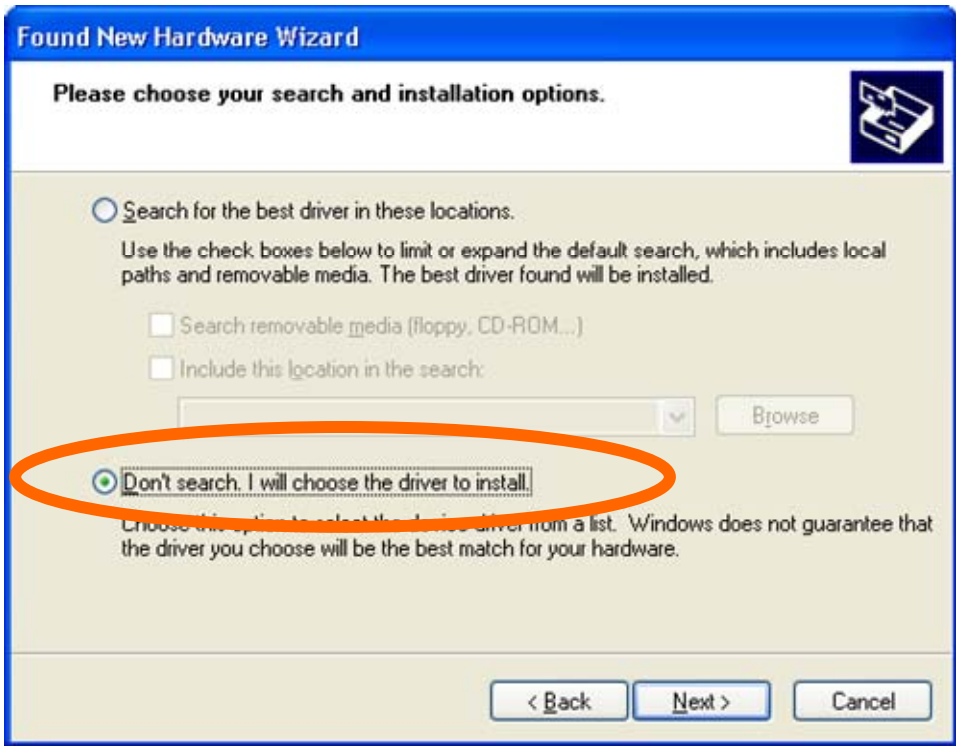
<1> When the connections of the TK-850/JG3H are recognized by the host, the Found New Hardware message is displayed, and then the Found New Hardware Wizard starts.

<2> On the first page of the **Found New Hardware Wizard** dialog box, select **No, not this time**, and then click the **Next** button.

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

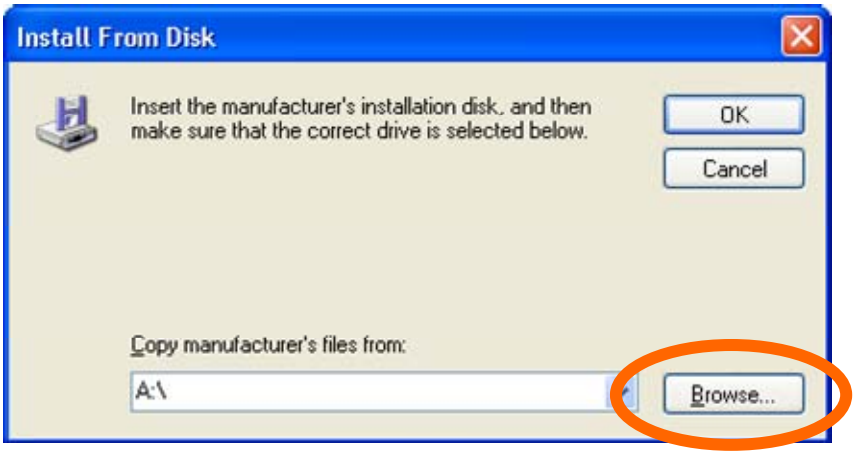


<4> On the next page, select **Don't search. I will chose the driver to install** and then click the **Next** button.

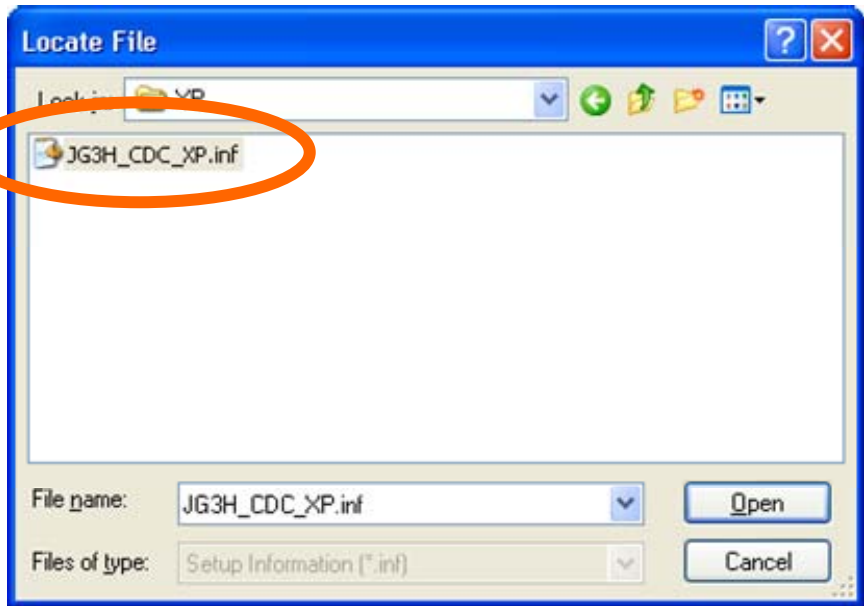


<5> On the next page, click the **Have Disk** button.

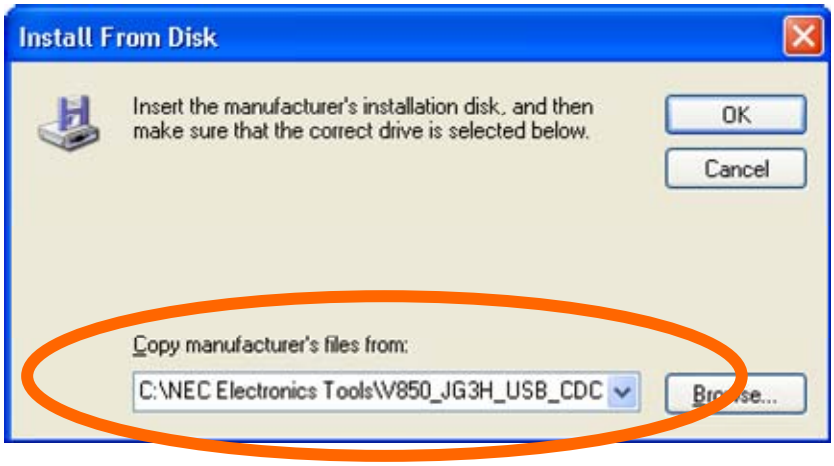
<6> In the **Install From Disk** dialog box, click the **Browse** button to display the `inf` file folder in the directory in which the sample driver was stored.



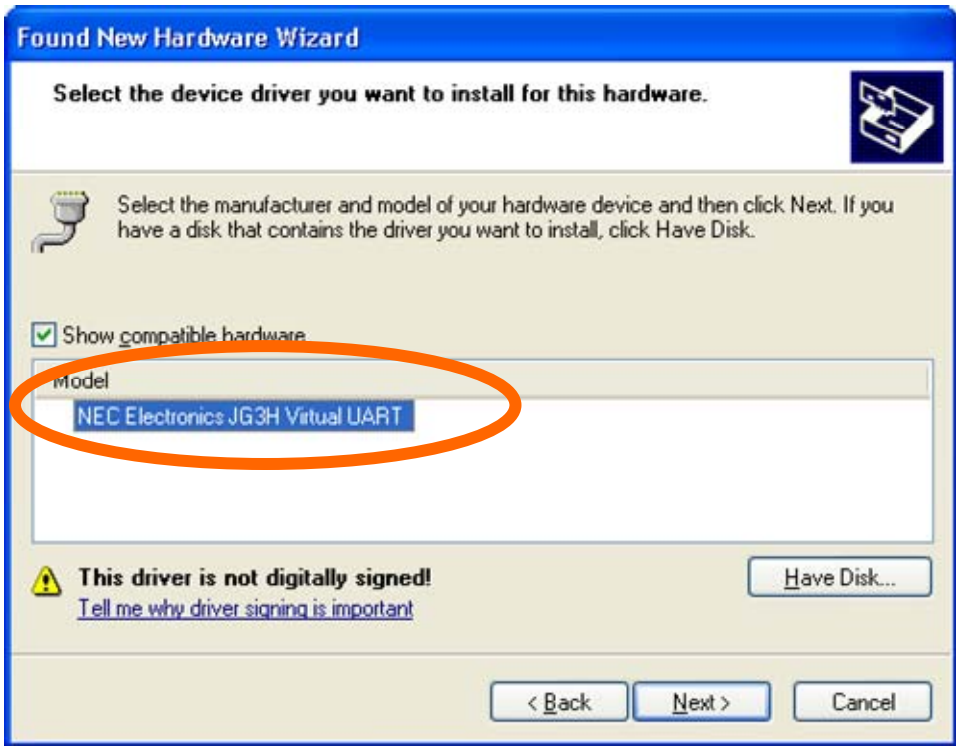
<7> Select the `inf` file in the **XP** or **VISTA** folder according to the OS used on the host, and then click the **Open** button.



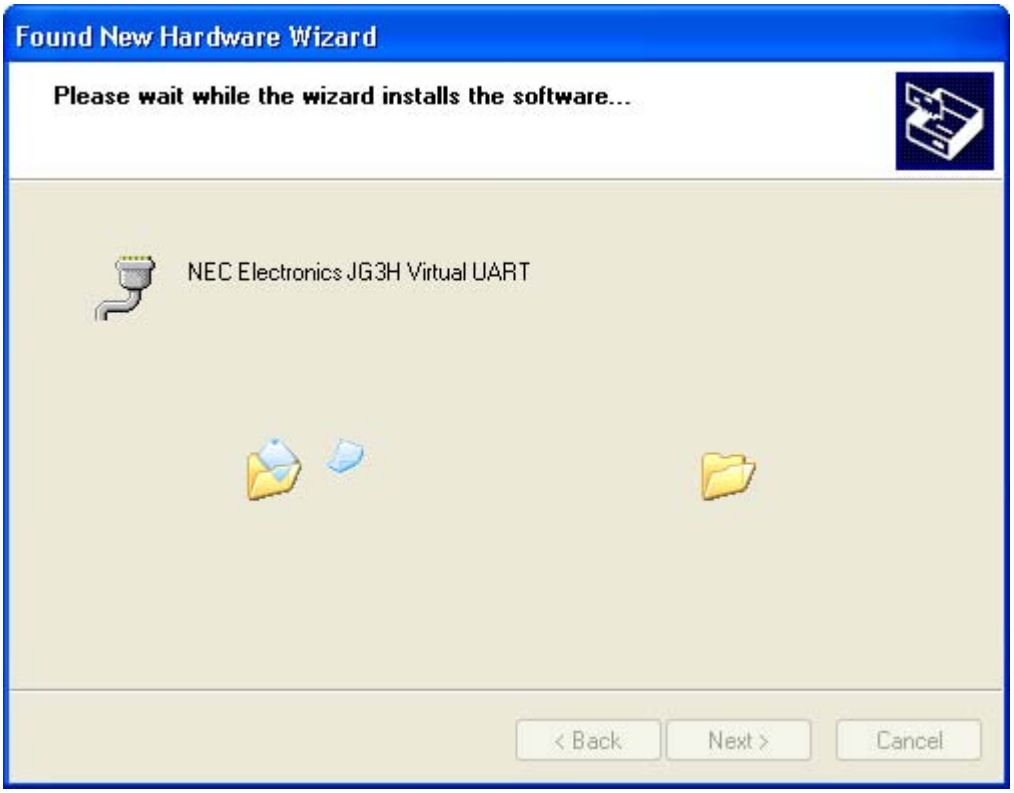
<8> In the **Install From Disk** dialog box, confirm that the path under **Copy manufacturer's files from:** is correct, and then click the **OK** button.



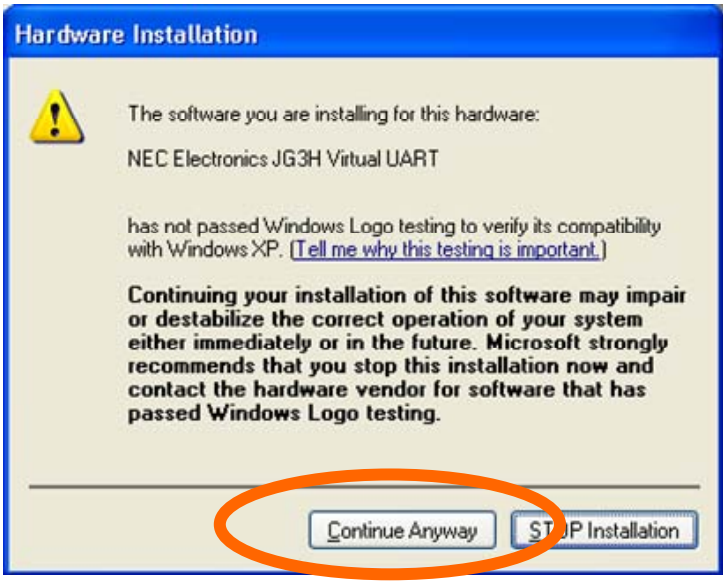
<9> In the **Found New Hardware Wizard** dialog box, select **NEC Electronics JG3H Virtual UART**, and then click the **Next** button.



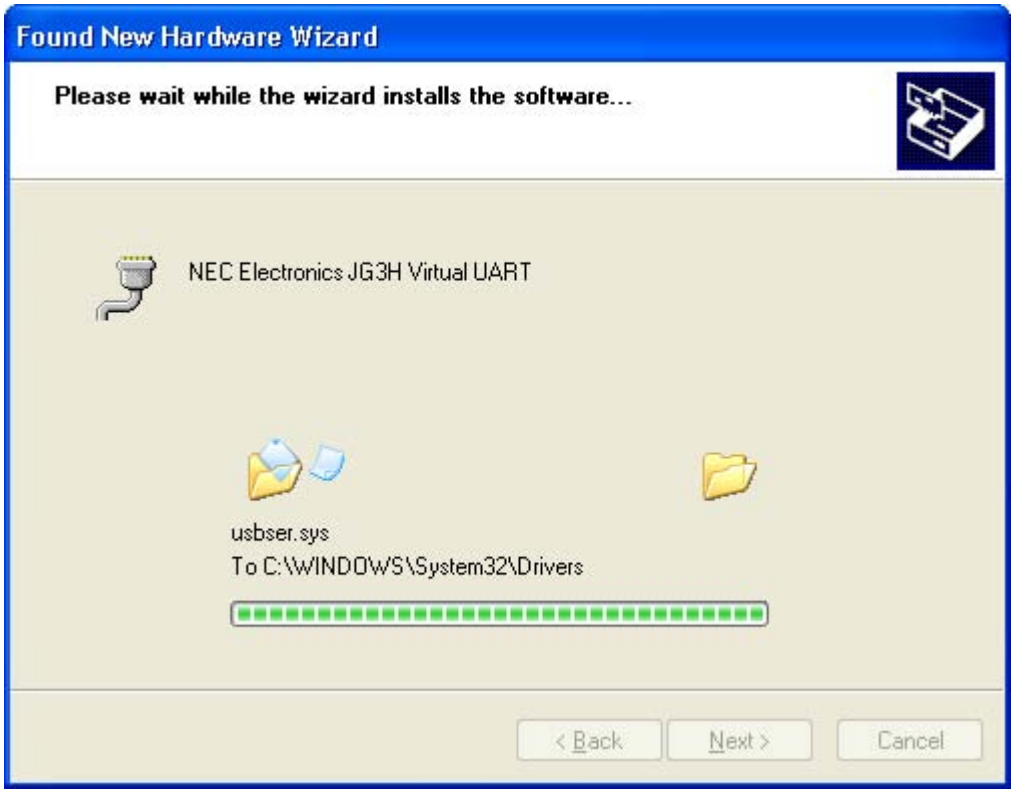
<10> The driver installation starts.



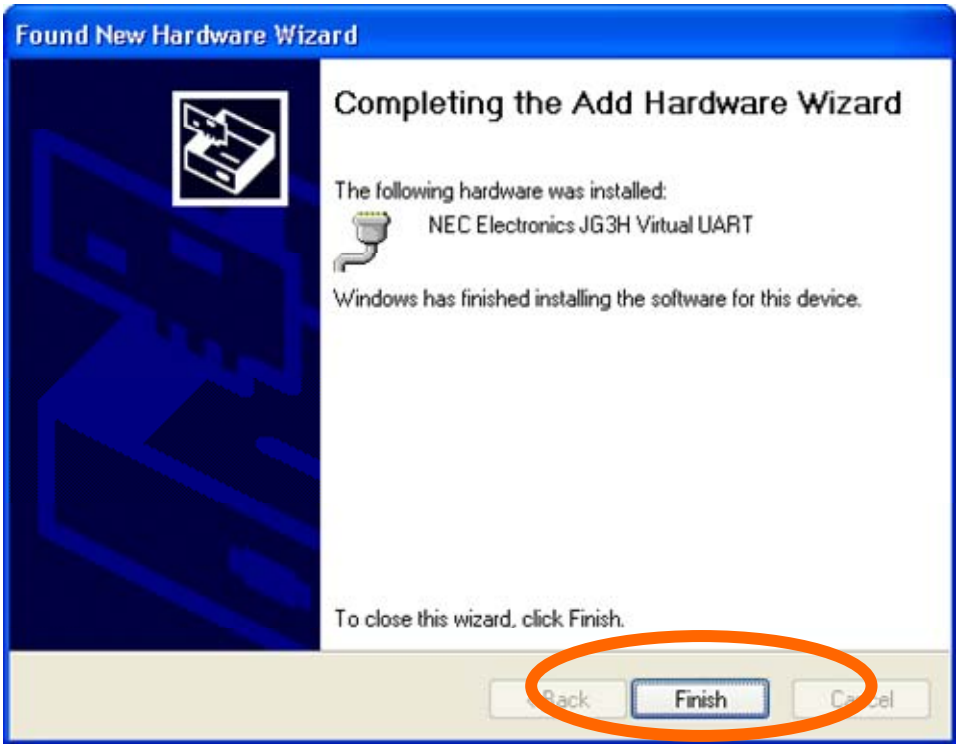
<11> In the **Hardware Installation** dialog box, click the **Continue Anyway** button.



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

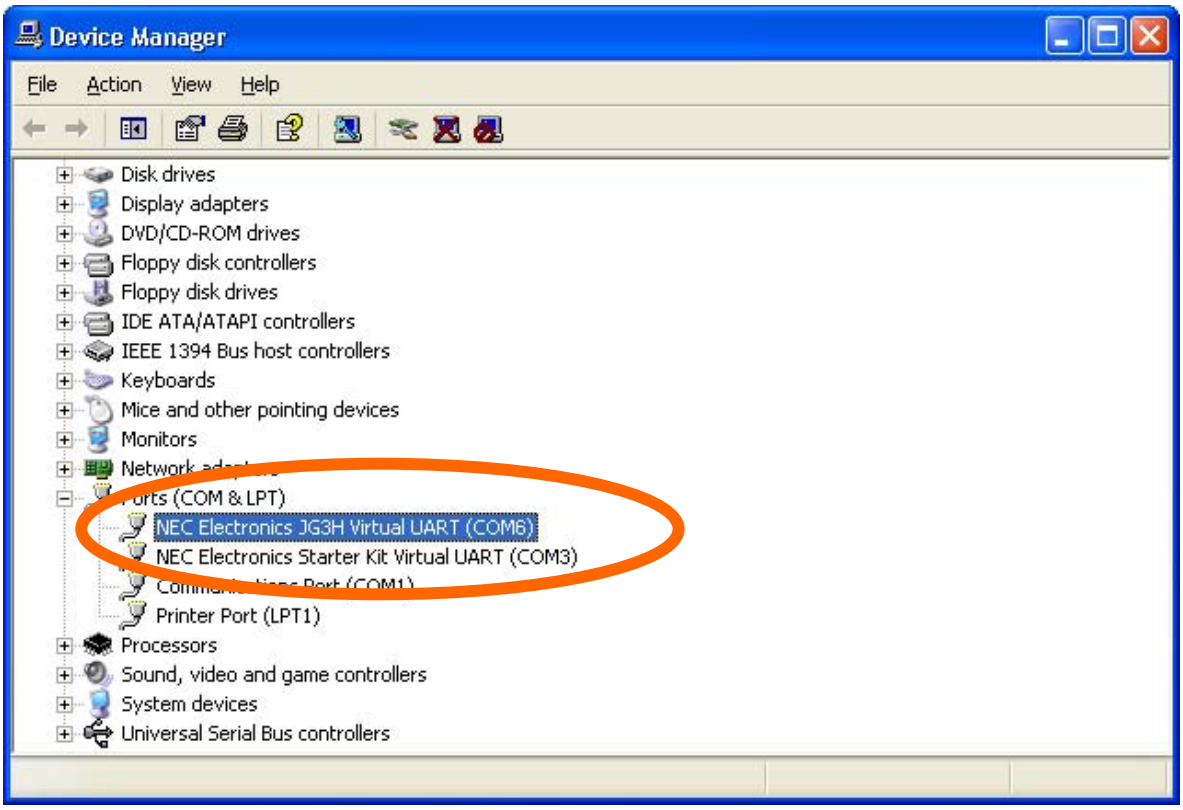


<13> On the next page, click the **Finish** button.



(3) Checking the device assignment

Open the Windows **Device Manager** window. In the **Ports (COM & LPT)** category, make sure that **NEC Electronics Jx3H Virtual UART** is displayed and check the assigned COM port number.



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

5.3 On-Chip Debugging

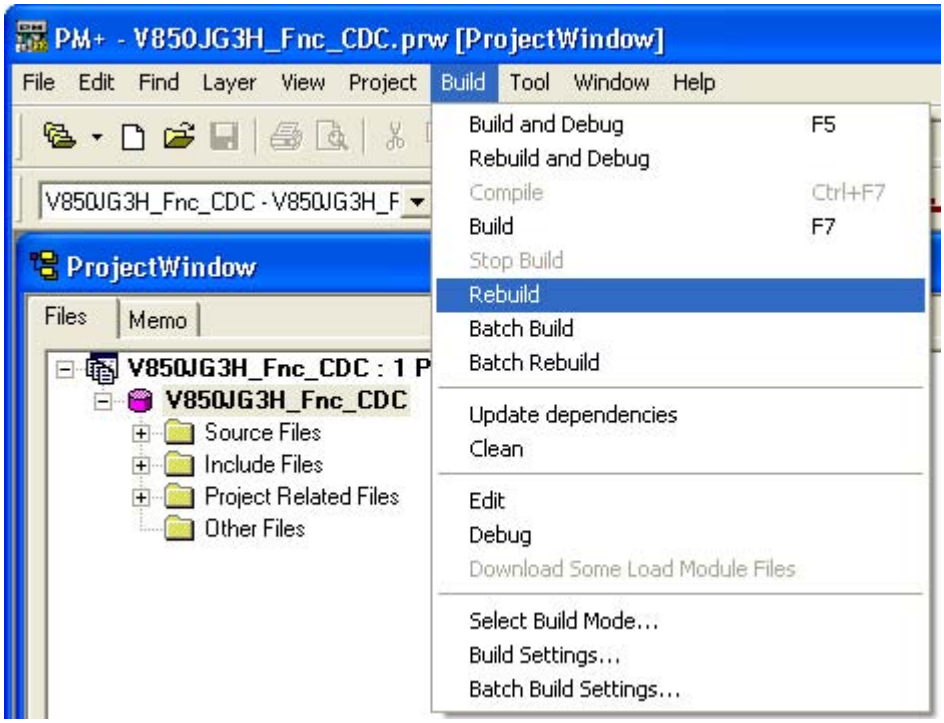
This section describes the procedure for debugging an application program that was developed using the workspace described in **5.2 Setting Up the Environment**.

For the V850ES/Jx3-H, 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.



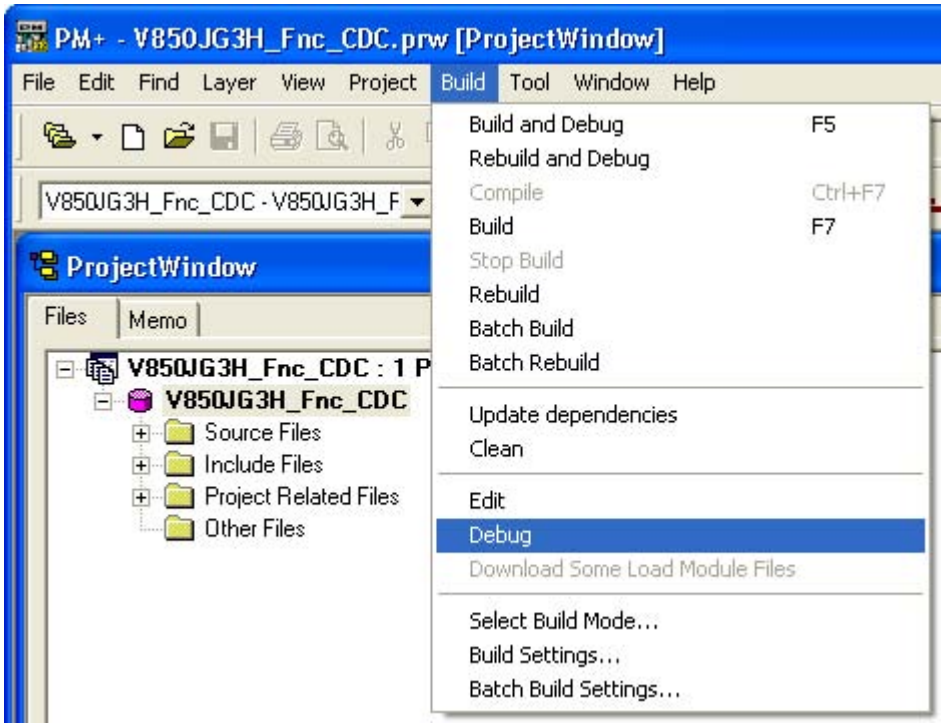
5.3.2 Loading and executing the load module

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

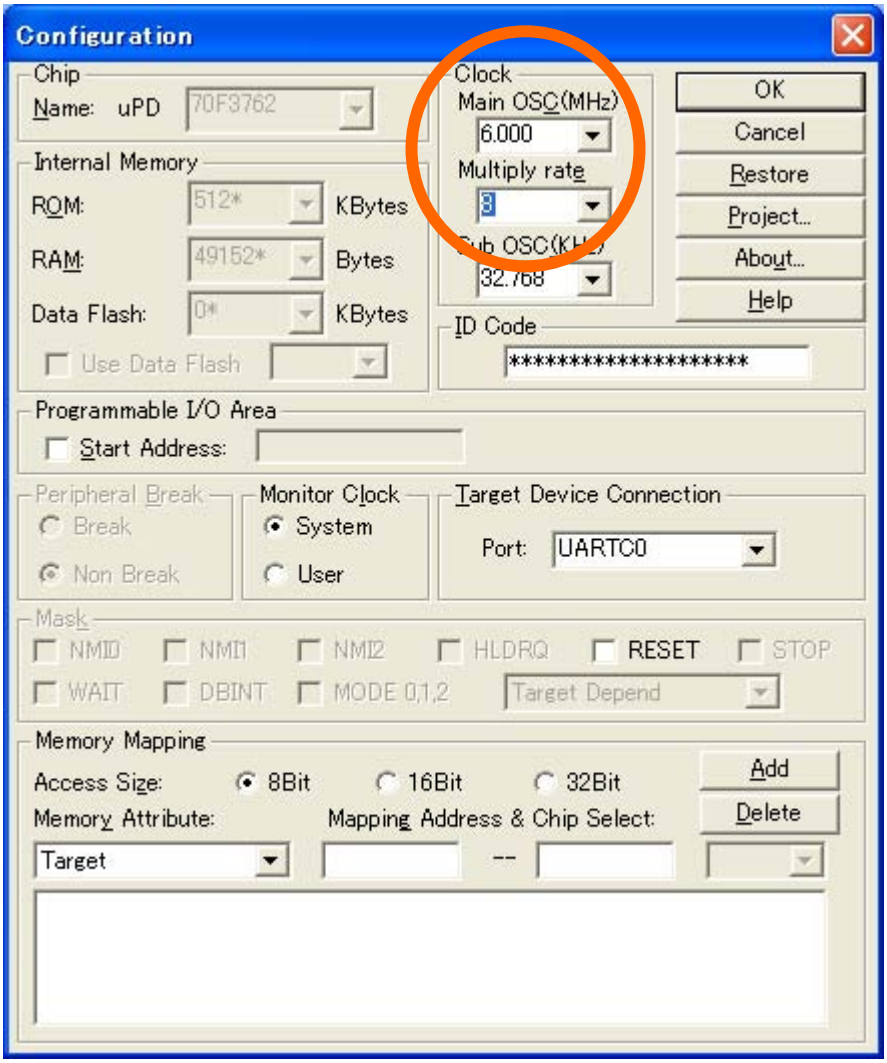
(1) Writing the load module

The procedure for writing the load module to the TK-850/JG3H by using PM+ is described below.

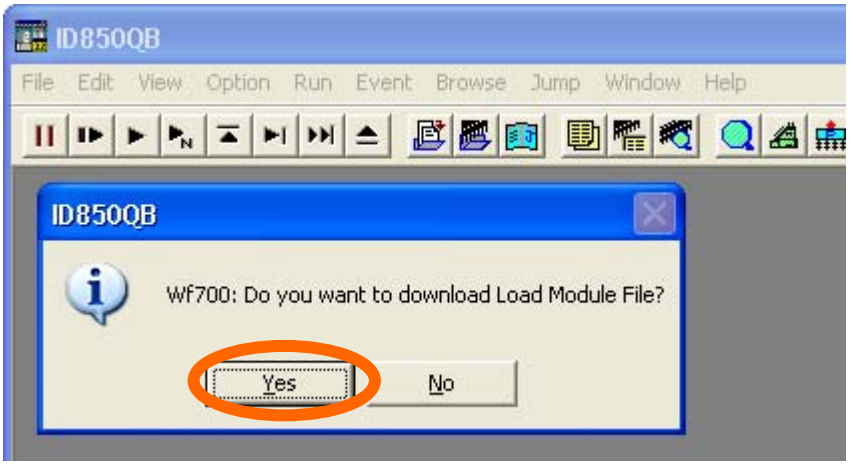
<1> Start the ID850QB by selecting **Debug** in the **Build** menu.



<2> In the **Configuration** dialog box, select “6.000” (MHz) for **Main OSC** and “8” for **Multiply rate**.

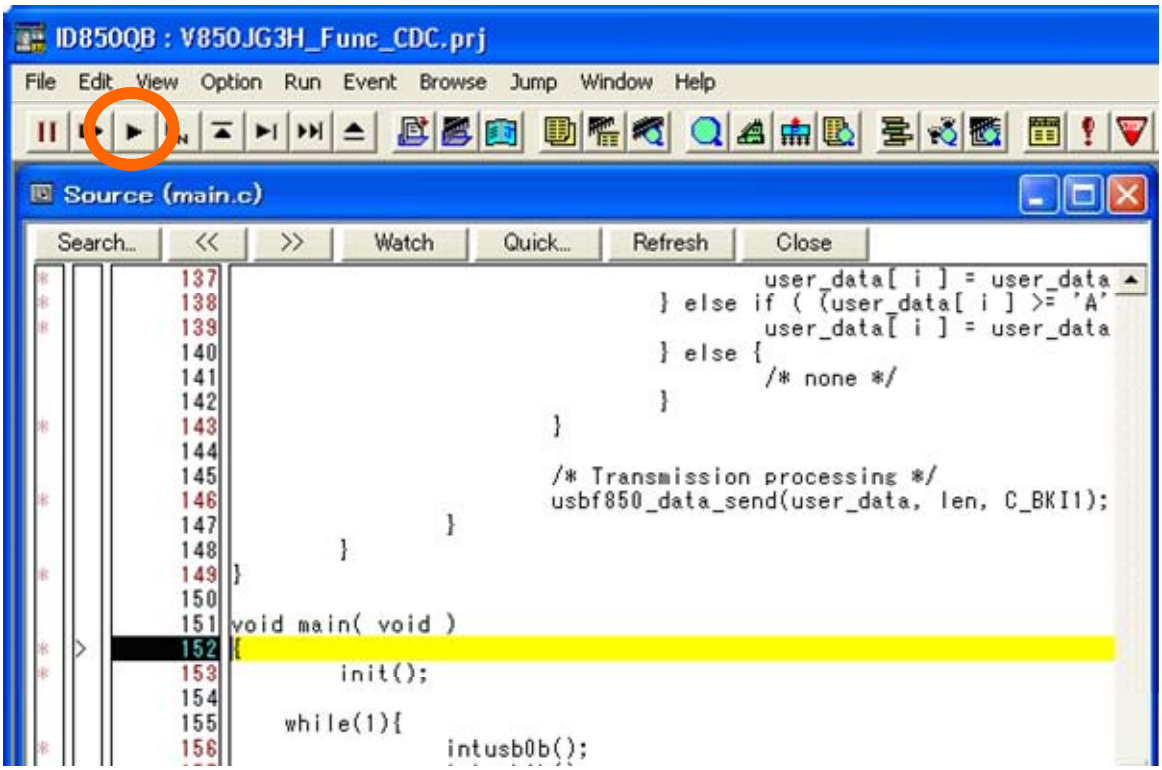


<3> If a project file included with the sample driver is used, the following dialog box is displayed. Click the **Yes** button to start writing the load module file.



(2) Executing the program

Click the ▶ button in the ID850QB window or select **Run Without Debugging** in the **Run** menu.



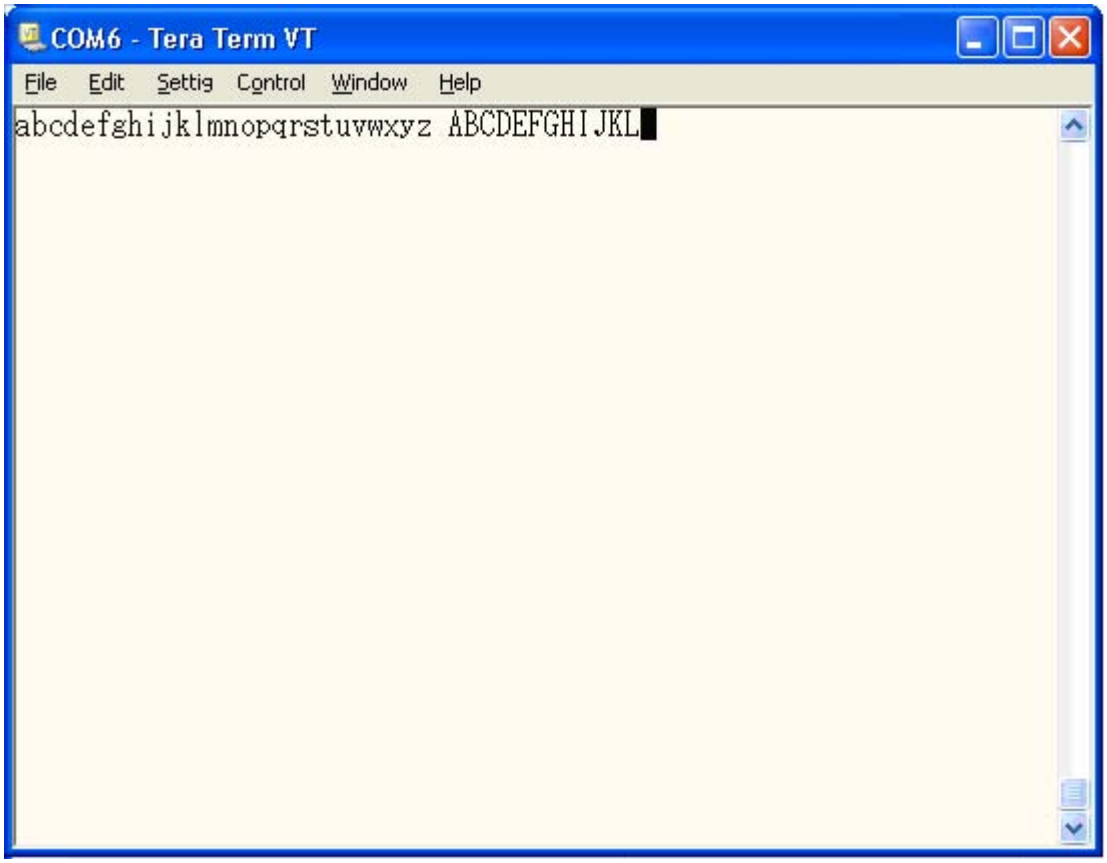
5.4 Checking the Operation

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:

- A character from "A" to "Z" should be displayed as "a" to "z".
- A character from "a" to "z" should be displayed as "A" to "Z".
- A character other than "A" to "Z" or "a" to "z" should be displayed as is.

Remark For details about the 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 CDC sample driver for the V850ES/Jx3-H.

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 registers in `RegDef.h` and `usbf850_sfr.h`
- The descriptor information in `usbf850_desc.h`
- 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` 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   Sample
3   void app_main( void )
4
5   Arguments:
6       N/A
7   Return values:
8       N/A
9   Overview:
10      sample application processing function.
11 =========================================================*/
12 void app_main( void )
13 {
14      UINT8 user_data[ USER_RBUF_SIZE ];
15      INT32 ret;
16      INT32 len;
17      INT32 i;
18
19      if ( usbf850_rdata_flg != 0 ) {
20           usbf850_rdata_flg = 0;
21
22      /* Receive processing */
23      memset( user_data , 0 , USER_RBUF_SIZE );
24      usbf850_rdata_length( &len , C_BKO1 );
25      ret = usbf850_data_receive(&user_data[ 0 ], len, C_BKO1);
26      if ( ret != DEV_ERROR ) {
27
28           /* uppercase <-> lowercase (ASCII code) */
29           for ( i = 0 ; i < len ; i++ ) {
30             if ( (user_data[ i ] >= 'a') && (user_data[ i ] <= 'z') ) {
31                user_data[ i ] = user_data[ i ] - 0x20;
32             } else if ( (user_data[ i ] >= 'A') && (user_data[ i ] <= 'Z') ) {
33                user_data[ i ] = user_data[ i ] + 0x20;
34             } else {
35                /* none */
36             }
37           }
38
39      /* Transmission processing */
40      usbf850_data_send(user_data, len, C_BKI1);
41           }
42      }
43 }
```

6.2.2 Setting up the registers

The registers the sample driver uses (writes to) and the values specified for them are defined in `RegDef.h` and `usbf850_sfr.h`. 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.

(1) RegDef.h

The CPU registers that are mainly used in initialization processing are defined in this file. (For details, see **3.2.1 CPU Initialization processing**.)

(2) usbf850_sfr.h

This file includes the definitions of the USBF registers, the register bits used in various types of processing, and the values specified for the bits. (For details, see **3.3.2 USBF initialization processing**.)

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 `usbf850_desc.h`. 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.

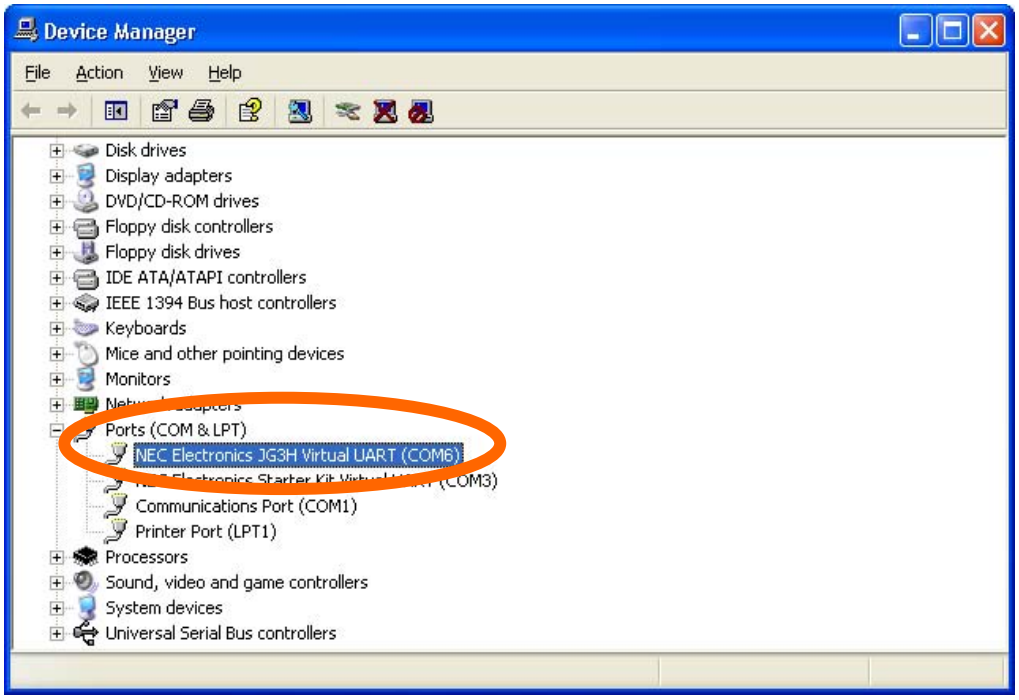
6.2.4 Setting up the virtual COM port host driver

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

(1) Changing the COM port number

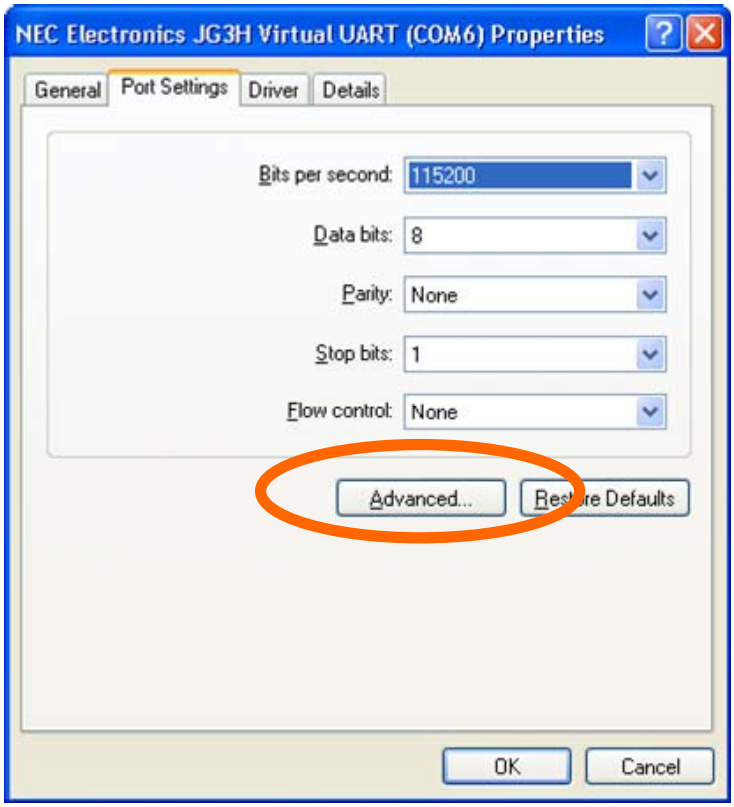
When the connection of a USB device is recognized by the host, the host automatically assigns the COM port number of the device, but the number can be changed to any number. To change the COM port number by using the host, perform the following procedure:

<1> Open the **Device Manger** window and display the items in the **Ports (COM & LPT)** category.

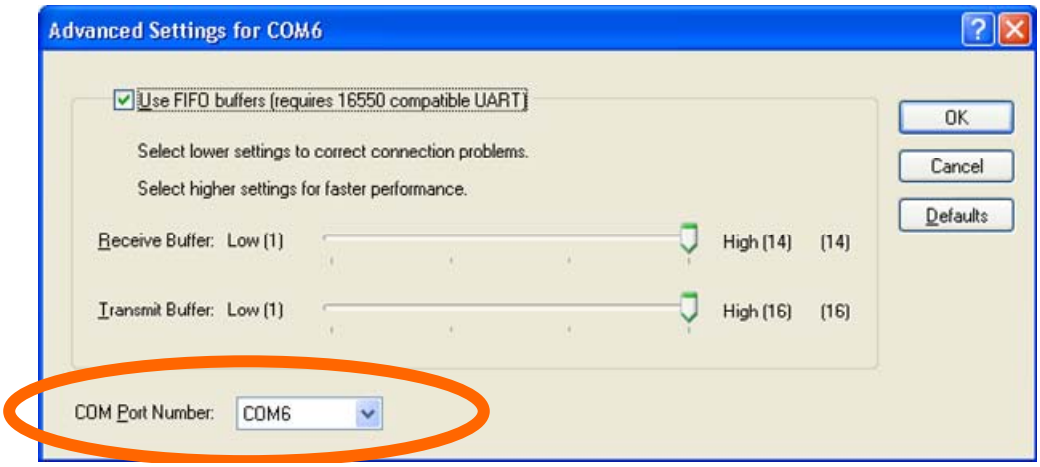


<2> Select **NEC Electronics Jx3H Virtual UART (COMn)** (where *n* is a number assigned by the host) to display its properties.

<3> Click the **Advanced** button on the **Port Settings** tab.



<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.



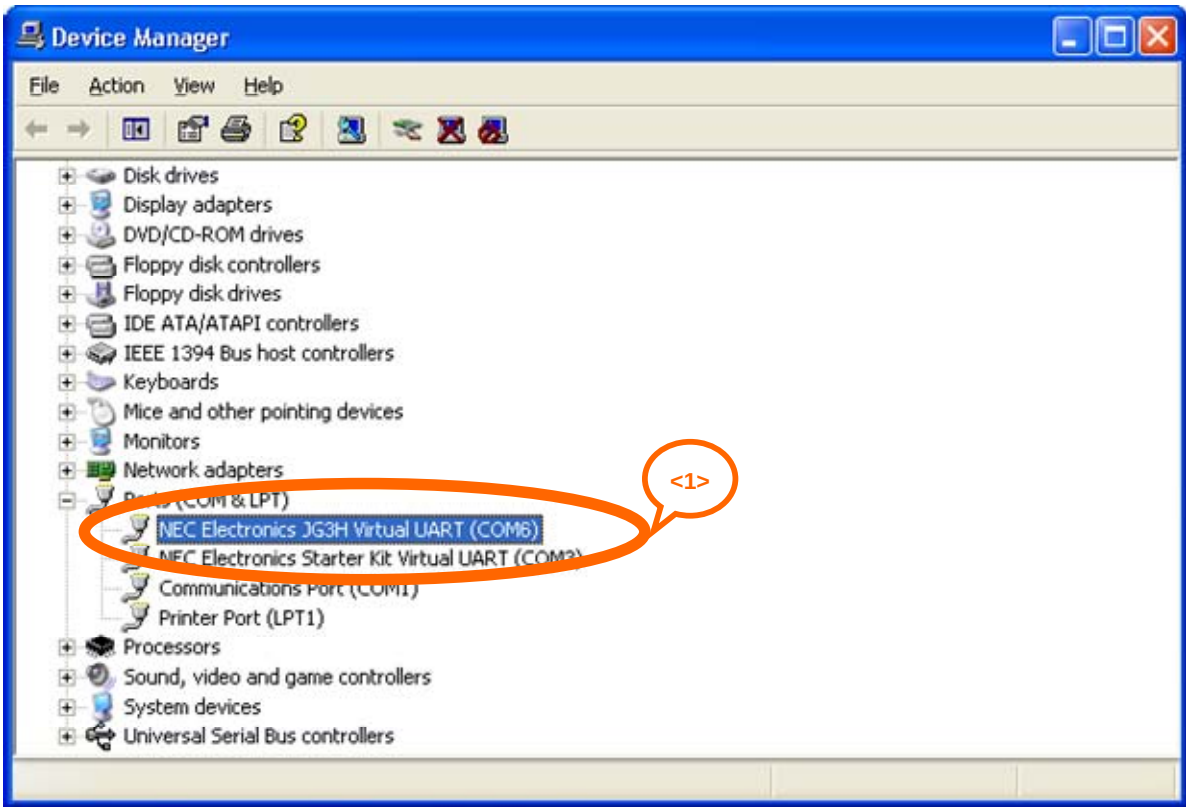
Remarks 1. Make sure not to select a port number that is used for a different device.

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

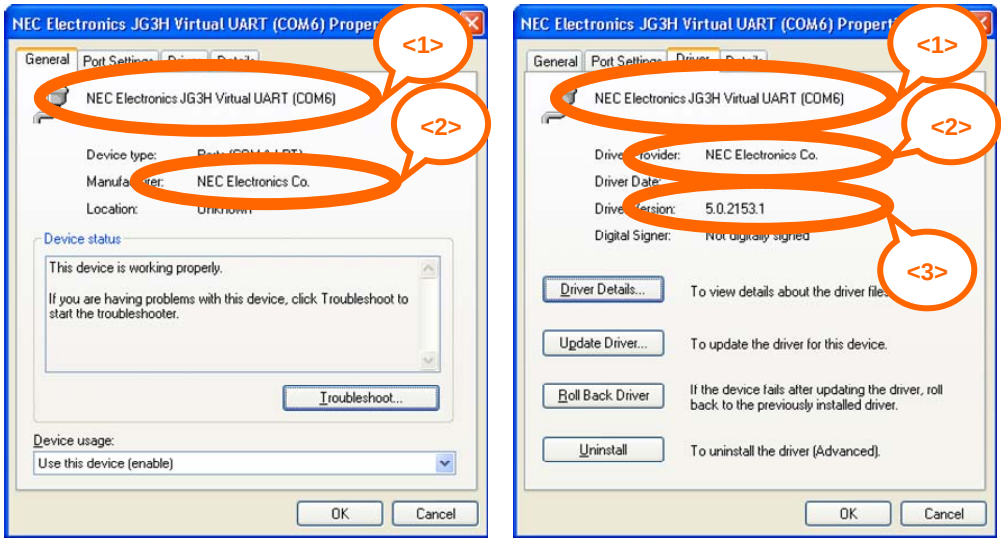
(2) Changing properties

Some information, such as the attributes of the device used by the 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)



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 6-2. INF File (`JG3H_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
7  Provider=%NEC%
8  LayoutFile=layout.inf
9  DriverVer=10/15/1999,5.0.2153.1                                    <3>
10
11 [Manufacturer]
12 %NEC%=NEC
13
14 [NEC]
15 %NECV850ESJx3H%=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, 0x00000002, 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
47 [Strings]
48 NEC = "NEC Electronics Corporation"                          <2>
49 NECV850ESJx3H = "NEC Electronics Jx3H Virtual UART"          <1>
50 Serial.SvcDesc = "USB Serial emulation driver"
```

(3) 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 6-1 can be reused as application examples for various types of defined processing.

(1) Judging whether data has been received

The data reception flag usbf850_rdata_flg is monitored on line 19. This flag is uniquely defined by the sample driver and the result of monitoring endpoint 2 for the sample driver (`intusb1b`) is applied to this flag. (For details, see **3.2.4 Monitoring endpoint 2**.) If this flag is set, it indicates that there is received data in the USB function controller.

(2) 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, and they are called on lines 24 and 25 (`usbf850_rdata_length` and `usbf850_data_receive`). The variables used as arguments are included on lines 14 to 17, and C_BKO1, which indicates the endpoint number, is defined in `usbf850_sfr.h`.

(3) Data transmission processing

The function that defines the processing for copying the transmitted data (`usbf850_data_send`) is called on line 40. The variables used as arguments are included on lines 14 to 17, and C_BKI1, which indicates the endpoint number, is defined in `usbf850_sfr.h`.

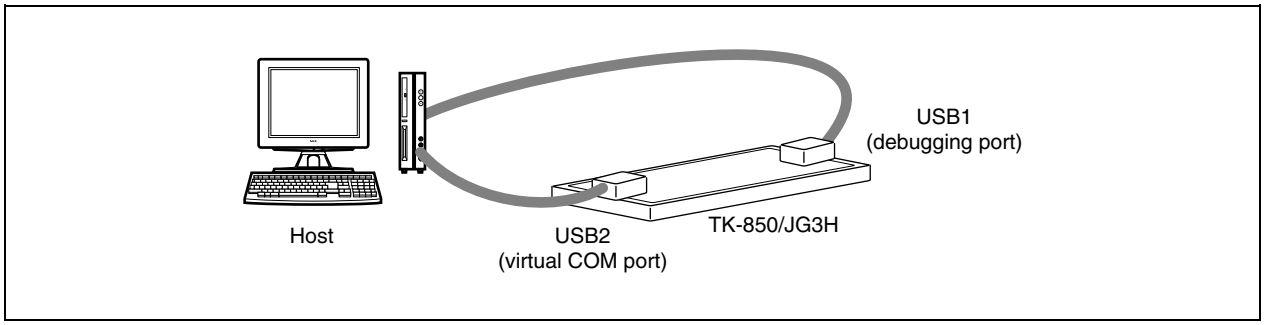
APPENDIX A STARTER KIT

This chapter describes the TK-850/JG3H starter kit for the V850ES/Jx3-H, made by Tessera Technology, Inc.

A.1 Overview

The TK-850/JG3H is a kit to develop applications that use the V850ES/Jx3-H. 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).

Figure A-1. Connections of the TK-850/JG3H



A.1.1 Features

The TK-850/JG3H has the following features:

- A USB miniB connector for the internal USBF
- Up to 84 I/O ports
- As small as a business card
- Efficient development by using the board with the integrated development environment (PM+)

A.2 Specifications

The main specifications of the TK-850/JG3H are as follows:

• CPU	μPD70F3760 (V850ES/JG3-H)
• Operating frequency	48 MHz (subsystem clock: 32.768 kHz)
• Interface	USB connector (miniB) × 2
	N-Wire connector (only the pad)
	MINICUBE®2 connector (SICA: only the pad)
	Peripheral board connector × 2 (only the pad)
• Supported platform	Host: DOS/V computer that has a USB interface
	OS: Windows 2000, Windows XP
• Operating voltage	5.0 V (internal operation at 3.3 V)
• Package dimensions	W89 × D52 (mm)

For further information,
please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
http://www.cn.necel.com/

Shanghai Branch
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai, P.R.China P.C:200120
Tel:021-5888-5400
http://www.cn.necel.com/

Shenzhen Branch
Unit 01, 39/F, Excellence Times Square Building,
No. 4068 Yi Tian Road, Futian District, Shenzhen,
P.R.China P.C:518048
Tel:0755-8282-9800
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
http://www.tw.necel.com/

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
http://www.kr.necel.com/

G0706