Application Note

78K0R/Kx3-L

(on-chip USB controller)

16-bit Single-Chip Microcontroller
USB HID (Human Interface Device) Class Driver

Renesas Electronics
www.renesas.com

R01AN0009ED0100, Rev. 1.00
July 12, 2010

Table of Contents

Chapter 1 Overview 4
1.1 Overview 4
1.1.1 Features of the USB function controller 4
1.1.2 Features of sample driver 5
1.1.3 Files included in the sample driver 5
1.2 Overview of 78K0R/Kx3-L 5
1.2.1 Applicable products 6
1.2.2 Features 6

Chapter 2 Overview of USB 8
2.1 Transfer Format 8
2.2 Endpoints 8
2.3 Device Class 9
2.4 Requests 9
2.4.1 Types 9
2.4.2 Format 10
2.5 Descriptor 11
2.5.1 Types 11
2.5.2 Format 12
2.5.3 HID class descriptor format 13

Chapter 3 Sample Driver Specification 15
3.1 Overview 15
3.1.1 Features 15
3.1.2 Supported requests 15
3.1.3 Descriptor settings 16
3.2 Operation of Each Section 19
3.2.1 CPU Initialization 20
3.2.2 USB function controller initialization processing 21
3.2.3 INTUSB interrupt process 24
3.3 Function Specification 26
3.3.1 Functions 26
3.3.2 Correlation of the functions 26
3.3.3 Function features 29

Chapter 4 Sample Application Specification 42
4.1 Overview 42
4.2 Operation 42
4.3 Using functions 43

Chapter 5 Development Environment 45
5.1 Development environment overview 45
5.1.1 Program development 45

5.1.2 Debugging 45
5.2 Setting up the Environment 46
5.2.1 Preparing Host Environment 46
5.2.2 Setting up the target environment 48
5.3 On-Chip Debugging....49
5.3.1 Generating the debug files 49
5.3.2 Download and Debug....49
5.4 Checking the Operation....51

Chapter 6 Using the Sample Driver....52
6.1 Overview 52
6.2 Customizing the sample driver 52
6.2.1 Application section....52
6.2.2 Setting up the device registers 53
6.2.3 Descriptor information 53
6.3 Using functions....53

Chapter 7 Starter Kit 54
7.1 Overview 54
7.1.1 Features....54
7.2 Specification 54

Chapter 1 Overview

This application note describes the USB HID (Human Interface Device) sample driver created for the USB function controller incorporated in the 78K0R/KC3-L, 78K0R/KE3-L (78K0R/Kx3-L) microcontrollers. This application note provides the following information.

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

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

1.1 Overview

1.1.1 Features of the USB function controller

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

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

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

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

- Automatically responds to standard USB requests (except some requests).
- Can operate as a bus-powered device or self-powered device[1]
- The internal or external clock can be selected[2]

[1] The sample driver selects bus power
[2] The sample driver selects the internal clock

1.1.2 Features of sample driver

The USB human interface device sample driver for the 78K0R/Kx3-L has the features shown below. For details about the features and operations, see Chapter 3 Sample Driver Specifications.

- Conforms to the USB human interface device class
- Operates as keyboard device
- Exclusively uses the following amounts of memory (excluding the vector table)
    - ROM: About 3.1 KB
    - RAM: About 0.4 KB

1.1.3 Files included in the sample driver

The sample driver includes the following files:

Table 1-2 Files included in the sample driver

Folder	File	Overview
src	main.c	Main routine, initialization, sample application
	usbf78k0r.c	USB initialization, endpoint control, bulk transfer, control transfer
	usbf78k0r_hid.c	Human interface device class specific processing
include	main.h	main.c function prototype declarations
	usbf78k0r.h	usbf78k0r. function prototype declarations
	usbf78k0r_hid.h	usbf78k0r_hid.c function prototype declarations
	usbf78k0r_desc.h	Descriptor definitions
	usbf78k0r_errno.h	Error code definitions
	usbf78k0r_types.h	User declarations

Remark In addition, the project-related files generated, when creating a development environment by using the IAR Embedded Workbench (an integrated development tool made by IAR Systems), are also included. For details see 5.2.1 Preparing the host environment.

1.2 Overview of 78K0R/Kx3-L

This section describes the 78K0R/KC3-L or 78K0R/KE3-L, which are controlled by using the sample driver.

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

1.2.1 Applicable products

The sample driver can be used for the following products.

Table 1-3 78K0R/Kx3-L Products

Generic Name	Part Number	Internal Memory		Incorporated USB Function	Interrupt	
		Flash Memory	RAM		Internal	External
78K0R/KC3-L (48pin)	μPD78F1022	64 KB	6 KB	Function controller	36	7
	μPD78F1023	96KB	8 KB	Function controller	36	7
	μPD78F1024	128KB	8 KB	Function controller	36	7
78K0R/KE3-L (64pin)	μPD78F1025	96KB	8 KB	Function controller	41	11
	μPD78F1026	128KB	8 KB	Function controller	41	11

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

1.2.2 Features

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

Memory space:

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

Internal memory

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

Multiplication/division function

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

Key interrupt

- 4 channels
- 8 channels

DMA controller

- 2 channels

Serial interface

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

USB controller

- USB function (full speed): 1 channel

A/D converter

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

Power supply voltage

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

Clock output/buzzer output

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

With built-in on chip debugging function

Note: Above mentioned information based on 78K0R/KE3-L

Chapter 2 Overview of USB

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

USB (Universal Serial Bus) is an interface standard for connecting various peripherals to a host system 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 aperiodically handle large amounts of data	Periodic data transfer format that has a low band width	Transfer format used for a real-time transfer
Specifiable packet size	High speed 480 Mbps	64 bytes	512 bytes	1 to 1,024 bytes	1 to 1,024 bytes
	Full speed 12 Mbps	8, 16, 32, or 64 bytes	8, 16, 32, or 64 bytes	1 to 64 bytes	1 to 1,023 bytes
	Low speed 1.5 Mbps	8 bytes	–	1 to 8 bytes	–
Transfer priority		3	3	2	1

2.2 Endpoints

An endpoint is an information unit that is used by the host device to specify a communicating device and is specified using a number from 0 to 15 and a direction (IN or OUT). An endpoint must be provided for every data communication path that is used for a peripheral device and cannot be shared by multiple communication paths[3]. 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).

[3] 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 human interface device (HID) class is intended for input devices connected to hosts, such as keyboard and mouse. Interface descriptor bInterfaceClass field is 0x03 for a HID class device. For details, see HID specifications (Device Class Definition for Human Interface Devices (HID) Specification Version 1.11).

2.4 Requests

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

2.4.1 Types

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

(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. It is a class request when bmRequestType field bit 6 value is 0 and bit 5 value is 1. For details, see HID specifications “Device Class Definition for Human Interface Devices (HID) Specification Version 1.11”.

Following requests are defined in HID class.

- **Get Report**

  This request is used to acquire data from function device using control transfer by the host. All drivers in compliance with HID class should support this request.

- **Get Idle**

  This request is used to acquire actual idle rate of function device by the host. Keyboard should support this request.

- **Get Protocol**
  This request is used to acquire existing protocol code of function device by the host. Boot device should support this request.

- **Set Report**
  This request is used to transmit data to function device by the host.

- **Set Idle**
  This request is used to set idle rate of function device by the host. Keyboard should support this request.

- **Set Protocol**
  This request is used to set protocol code of function device by the host. Boot device should support this request.

(3) Vendor request

Vendor requests are the request defined unique to vendor. Host driver responding to this request should be provided to vendor while using vendor request. It is vendor request when bmRequestTypefield bit 6 value is 1 and bit 5 value is 0.

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
7		Higher	(the data length)

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

2.5.3 HID class descriptor format

The format of HID class descriptor report descriptor is as follows.

Table 2-9 HID Descriptor Format

Field	Size (Bytes)	Description
bLength	1	Descriptor size (0x09 fixed)
bDescriptorType	1	Descriptor type (0x21 fixed)
bcdHID	2	HID version (BCD expression)
bCountryCode	1	Country code number
bNumDescriptors	1	Number of class descriptor
bDescriptorType	1	Class descriptor type (first)
wDescriptorLength	2	Class descriptor size (first)
[bDescriptorType]...	1	Class descriptor size (from second no. onwards) (optional)
[wDescriptorLength]...	2	Class descriptor size (from second no. onwards) (optional)

Table 2-10 Report Descriptor Format

Short item

Bit	39/23/15 8	7 6 5 4	3 2	1 0
Item	[data] (0~3byte)	bTag	bType	bSize

Long item

Bit	2041 24	23 ... 16	15 ... 8	7 6 5 4	3 2	1 0
Item	[data] (0~255 byte)	bLongItemTag	bDataSize	1 1 1 1	1 1	1 0

Field	Description
bTag	4 bit field to specify item contents. Used in combination with bType. 111 fixed at long item.
bType	2 bit field to specify item type. There are three types 00: main, 01: global, 10: local. 11 fixed at long item.
bSize	2 bit field to specify data (data field) size by byte. 10(2 byte) fixed at long item.
bLongItemTag	8 bit field to specify item contents while using long item.
bDataSize	8 bit field to specify data (data field) size by byte while using long item.

Table 2-11 Main Item (bType = 0x00) Format

bTag	Tag name	Data	
		Bit	Description
1000	Input	Specifies input data format.	
		b31-b9	Reserved(0 fixed)
		b8	0:bit unit field, 1:Byte unit buffer
		b7	Reserved(0 fixed)
		b6	0:Null data disable, 1:Null data enable
		b5	0:priority, 1:Nonpriority
		b4	0:Linear, 1:Non-linear
		b3	0:No roll over, 1:Roll over
		b2	0: Absolute value, 1:Relative value
		b1	0: Array, 1: Variable
		b0	0:Data, 1: Constant
1001	Output	Specifies output data format.	
		b31-b8	Same as Input(bTag = 1000)
		b7	0: Value without changes, 1:Changed value
		b6-b0	Same as Input(bTag = 1000)
1011	Feature	Specifies device composition information.	
		b31-b0	Same as Output(bTag = 1001)
1010	Collection	Specifies data (Input, Output, and Feature) set. 0x00:Physical(coordinates) 0x01:Application(Such as mouse and keyboard) 0x02:Logical(Interrupt data) 0x03:Report 0x04:Named Array 0x05:Usage Switch 0x06:Usage Modifier 0x07-0x7F:Reserved 0x80-0xFF:Vendor-defined	
1100	End Collection	Specifies Collection termination. Does not contain data (bSize = 0).	

Table 2-12 Global Item (bType = 0x01) Format

bTag	Tag name	Data
0000	Usage Page	Item ID number
0001	Logical Minimum	Minimum value of variable and array
0010	Logical Maximum	Maximum value of variable and array
0011	Physical Minimum	Minimum physical limit of changeable item
0100	Physical Maximum	Maximum physical limit of changeable item
0101	Unit Exponent	Exponent at cardinal number 10 (2's complement)
0110	Unit	Unit
0111	Report Size	Each report size (bit unit)
1000	Report ID	Report ID number
1001	Report Count	Number of reports
1010	Push	Storing of global item status list in stack.
1011	Pop	Retrieval of global item status list stored at the starting of stack.

Table 2-13 Local Item (bType = 0x02) Format

bTag	Tag name	Data
0000	Usage	Item ID number
0001	Usage Minimum	Starting position of array and bitmap
0010	Usage Maximum	Ending position of array and bitmap
0011	Designator Index	Physical descriptor ID
0100	Designator Minimum	Starting position of identification information related to array and bitmap
0101	Designator Maximum	Ending position of identification information related to array and bitmap
0111	String Index	String descriptor ID
1000	String Minimum	First (starting) ID at the time of multiple string descriptors of array and bitmap
1001	String Maximum	Last (ending) ID at the time of multiple string descriptors of array and bitmap
1010	Delimiter	1:Start of local item, 0: End of local item

Chapter 3 Sample Driver Specification

This chapter provides details about the features and processing of the USB human interface device class sample driver for the 78K0R/Kx3-L and the specifications of the functions provided in the 78K0R/Kx3-L.

3.1 Overview

3.1.1 Features

The sample driver can perform the following processing.

(1) Initialization

The USB function controller is set up for use by manipulating various registers. This setup includes specifying settings for the CPU registers of the 78K0R/Kx3-L and specifying settings for the registers of the USB function controller. For details, see 3.2.1 CPU Initialization, 3.2.2 USB function controller initialization processing.

(2) Monitoring endpoints

The status of transfer endpoints in USB function controller is notified from INTUSB interrupt. There are CPUDEC interrupt expressing the request of decode by FW for the control transfer endpoint (Endpoint0). During the processing of Endpoint0, requests are responded too .For details, see 3.2.3 INTUSB interrupt process. Key data is transmitted in endpoint (Endpoint7) for interrupt transfer.

(3) Sample application

It is operated as HID keyboard device. Key interrupt is generated by pressing SW and key data is transmitted.

3.1.2 Supported requests

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

(1) Standard requests

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

(a) GET_DESCRIPTOR_string

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

(b) Other requests

The sample driver returns a STALL.

(2) Class requests

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

(a) Get Report

This request is used to acquire data from HID device using control transfer by the host. If this request is received, sample driver transmits stored key code.

(b) Get Idle

This request is used to acquire current idle rate of function device by the host. If this request is received, sample driver transmits the current idle rate (=0).

(c) Set Idle

This request is used to acquire current idle rate of function device by the host. Sample driver supports only "0" idle rate. Sample driver returns NULL response when idle rate specified by this request is "0". It returns a STALL in case of other than "0" idle rate.

(d) Get Protocol, Set Report, Set Protocol

Sample driver does not support this request. If this request is received, sample driver transmits a STALL.

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 "usbhid_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: HID
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	0x01D9	Product ID:78K0R /Kx3-L
bcdDevice	2	0x0001	Device release number:1st version
iManufacturer	1	0x01	Index to the string descriptor representing the manufacturer: 1
iProduct	1	0x02	Index to the string descriptor representing the product: 2
iSerialNumber	1	0x03	Index to the string descriptor representing the device production number:3
bNumConfigurations	1	0x01	Number of configurations:1

(2) Configuration descriptor

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

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	0x0022	Total number of bytes of the configuration, interface, and endpoint descriptors: 34 bytes
bNumInterfaces	1	0x01	Number of interfaces in this configuration: 1
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	0xA0	Features of this configuration: bus-powered, with 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 USB function controller is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request.

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	0x03	Class code: communications interface class
bInterfaceSubClass	1	0x00	Subclass code: Abstract Control Model
bInterfaceProtocol	1	0x01	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 USB function controller is initialized, because the hardware automatically responds to a GET_DESCRIPTOR_configuration request.

Table 3-4 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	0x0040	Maximum packet size of this transfer: 64 bytes
bInterval	1	0x00	Polling interval of this endpoint: 0 ms

(5) String descriptor

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

Table 3-5 String 0 Descriptor Settings

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.)

Table 3-6 String 1 Descriptor Settings

Field	Size (Bytes)	Specified Value	Description
bLength[4]	1	0x2A	Descriptor size: 42 bytes
bDescriptorType	1	0x03	Descriptor type: string
bString[5]	40	-	Vendor: NEC Electronics Corporation

Table 3-7 String 2 Descriptor Settings

Field	Size (Bytes)	Specified Value	Description
bLength[4]	1	0x0E	Descriptor size: 14 bytes
bDescriptorType	1	0x03	Descriptor type: string
bString[5]	12	-	Product type: HID Drv(HID driver)

Table 3-8 String 3 Descriptor Settings

Field	Size (Bytes)	Specified Value	Description
bLength[4]	1	0x18	Descriptor size: 22 bytes
bDescriptorType	1	0x03	Descriptor type: string
bString[5]	20	-	Serial number: 01D903000110

(6) HID descriptor

HID (Human Interface Device) descriptor is used to define number and format of report descriptor and physical descriptor. Sample driver transmits HID descriptor from control endpoint if GET_DESCRIPTOR_HID request is received.

Table 3-9 Settings for HID descriptor

Field	Size (Bytes)	Specified Value	Description
bLength	1	0x09	Descriptor size: 9 bytes
bDescriptorType	1	0x21	Descriptor type: HID
bcdHID	2	0x0110	HID version (BCD expression)
bCountryCode	1	0x00	No country code number
bNumDescriptors	1	0x01	Number of class descriptor :1
bDescriptorType	1	0x22	Class subordinate descriptor type : HID report
wDescriptorLength	2	0x002E	Class subordinate descriptor length :46Byte

[4] The specified value depends on the size of the bString field.
[5] The vendor can freely set up the size and specified value of this field

(7) Report descriptor

Report descriptor is used to define format (report protocol) of the data (HID data) transmitted/received in between the host and function device. Sample driver transmits this descriptor from control endpoint in response to GET_DESCRIPTOR_report request. For details of each item, see **Universal Serial Bus HID Usage Tables Version 1.12.**

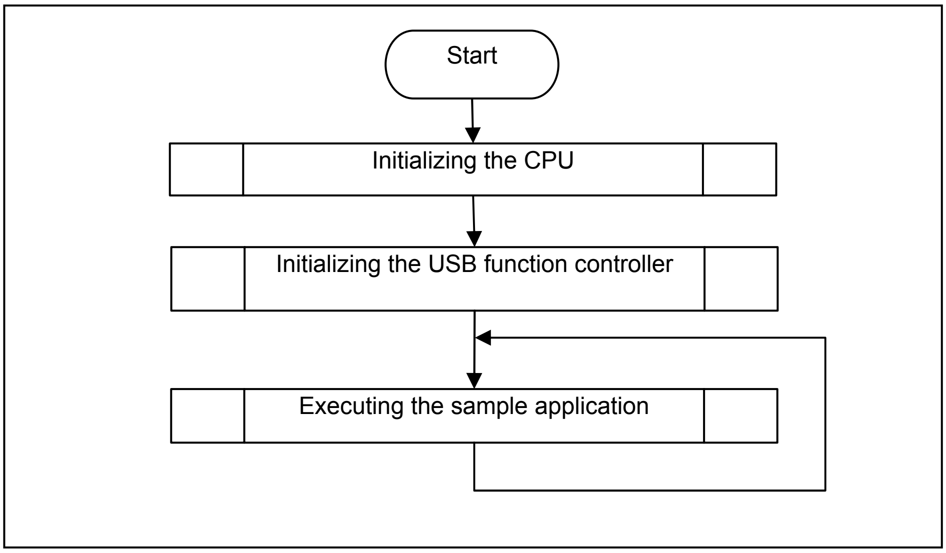
Table 3-10 Settings for report descriptor

Value	Item	Settings
0x05, 0x01	Usage Page	Generic Desktop
0x09, 0x06	Usage	Keyboard
0xA1, 0x01	Collection	Application
0x05, 0x07	Usage Page	Keyboard
0x19, 0xE0	Usage Minimum	LEFT CTRL
0x29, 0xE7	Usage Maximum	RIGHT GUI
0x15, 0x00	Logical Minimum	0
0x25, 0x01	Logical Maximum	1
0x95, 0x08	Report Size	8 bit
0x75, 0x01	Report Count	1
0x81, 0x02	Input	Variable
0x95, 0x01	Report Count	1
0x75, 0x08	Report Size	8 bit
0x81, 0x01	Input	Constant
0x95, 0x06	Report Count	6
0x75, 0x08	Report Size	8 bit
0x15, 0x00	Logical Minimum	0
0x26, 0xFF, 0x00	Logical Maximum	255
0x05, 0x07	Usage Page	Keyboard
0x19, 0x00	Usage Minimum	0
0x29, 0x91	Usage Maximum	145
0x81, 0x00	Input	-
0xC0	End Collection	-

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, please refer to Chapter 4 Sample Application Specifications.

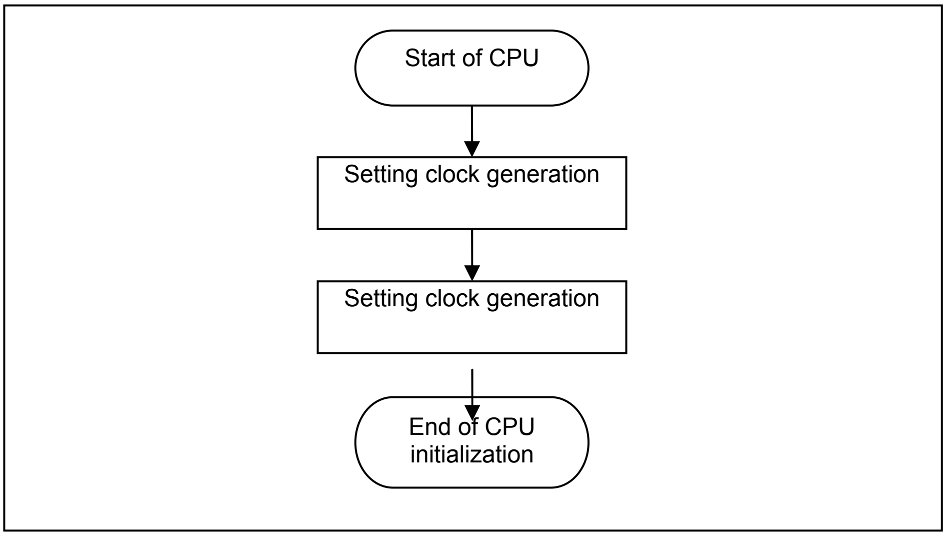
Figure 3-1 Sample Driver Processing Flowchart



3.2.1 CPU Initialization

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

Figure 3-2 CPU Initialization Flowchart



(1) Clock generation settings

Operation of internal clock of CPU is set. Here, five registers are set.

(a) “0x41” is written to CMC register to specify X1 oscillation mode, 10MHz < f_{MX} <= 20MHz.

(b) “0” is written to the MSTOP bit of CSC register to start the operation of X1 oscillation circuit.

(c) Oscillation stability time is verified according to OSTC register.

(d) “0x01” is written in PLLC register to stop the PLL operation.

(e) “0x38” is written to the CKC register to specify CPU/peripheral hardware clock to main system clock (f_{MAIN}), main system clock to high speed system clock (f_{MX}) and ratio of dividing frequency to f_{MX} .

(f) “1” is written to the HIPSTOP bit of CSC register to stop high speed built-in oscillation circuit.

(g) “1” is written to PLLM bit of PLLC register to multiply the frequency of the clock provided to PLL by 12.

(h) “0” is written to PLLSTOP bit of PLLC register to stat the operation of PLL.

(2) Port Settings

(a) Pull-up option of port connecting to switch is set. “1” is written to P17, P43, P42, P70, P71, P72, P73, and P74.

(b) Processing should be done after reset release in the unused pin to avoid flow of penetration current if regular Schmidt input type pin is open. In sample driver (64pin edition settings) “0” is written to P05,P06,P30,P44,P46,P47,P54,P55,P65,P66,P67.

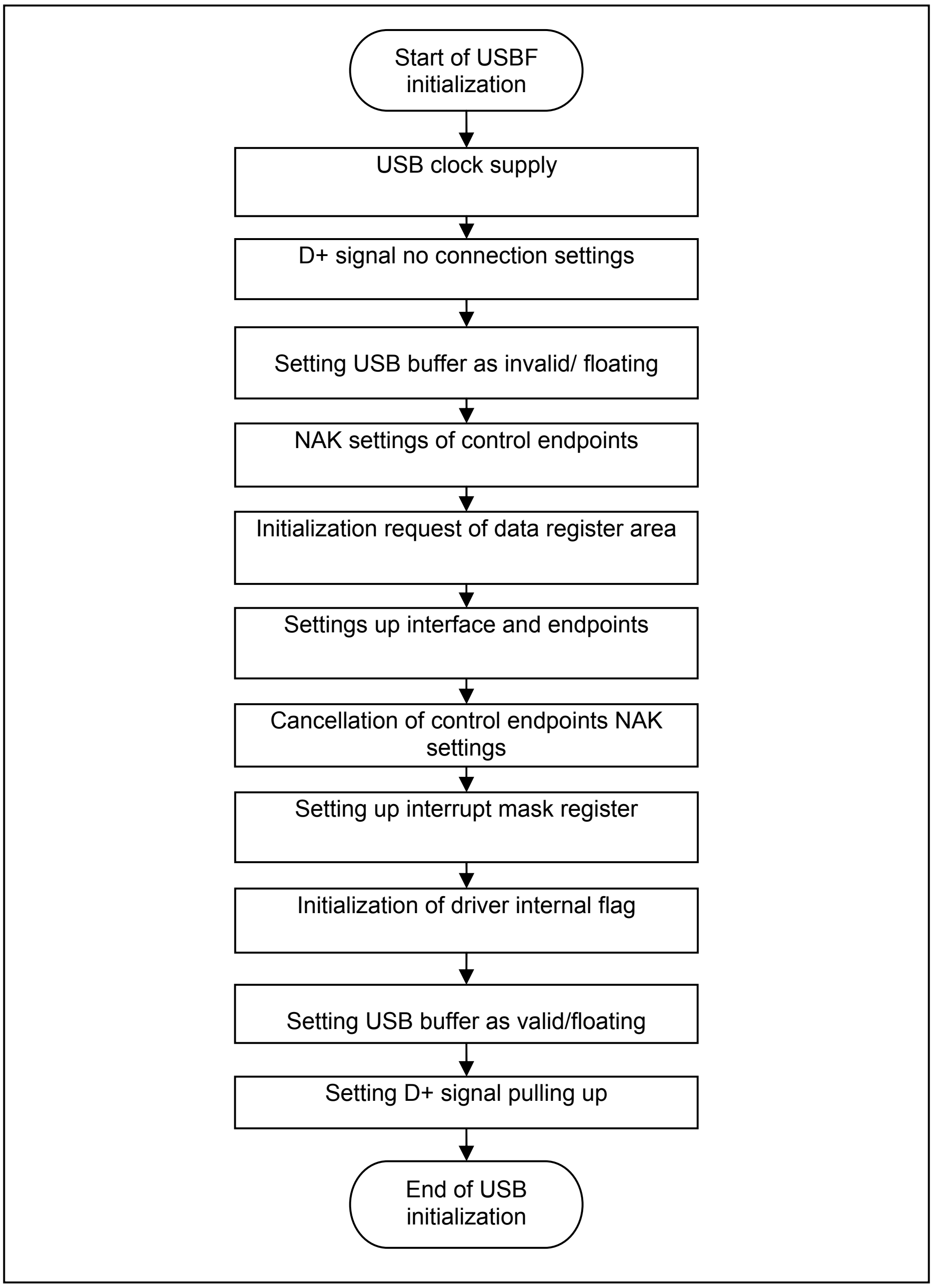
Note In case of 48pin, addition to the above mentioned pins, processing should be done in P42,P43,P53,P74,P75,P77,P110,P142,P143 also.

(c) LED settings. "0" is written to the PM2, PM5, PM110, and PM110. "0xff" is written to P2, P5 to turn off the LED. "1" is written to P110 after writing "0" to it to turn OFF LED (7SegA). Next, ""0xff" is written to P2, P5 again and "1" is written to P111 after writing "0" to it to turn OFF LED (7SegB).

3.2.2 USB function controller initialization processing

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

Figure 3-3 USB function controller Initialization Processing Flowchart



(1) USB clock supply

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

(2) D+ Signal no-connection settings

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

(3) Invalidate USB buffer and validate the floating measures

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

(4) NAK settings of control endpoints

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

(5) Initializing the request data register area

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

(a) "0x02" is written to the UF0DSTL register to disable remote wakeup and operate the USB function controller as a bus-powered device.

(b) "0x00" is written to the UF0EnSL registers (where n = 0 to 7) to indicate that endpoint n operates normally.

(c) The total data length (number of bytes) of the required descriptor is written to the UF0DSCL register to determine the range of the UF0CIEn registers (where n = 0 to 255).

(d) The device descriptor data is written to the UF0DDn registers (where n = 0 to 7).

(e) The data of the configuration, interface, and endpoint descriptors is written to the UF0CIEn registers (where n = 0 to 255).

(f) "0x00" is written to the UF0MODC register to enable automatic responses to GET_DESCRIPTOR_configuration requests.

(6) NAK settings of interface and endpoints

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

(a) "0x00" is written to the UF0AIFN register to enable one interface.

(b) "0x00" is written to the UF0AAS register to disable the alternative setting.

(c) "0x20" is written to the UF0E7IM register to link endpoint 7 to interface 0.

(7) Disabling NAK settings of control endpoints

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

(8) Setting up the interrupt mask registers

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

(a) "0x00" is written to the 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) "0x3B" is written to the UF0IM0 register to mask all interrupt sources other than BUSRST, RSUSPDM, SETRQ interrupts from the interrupt sources indicated by the UF0IS0 register.

(d) "0x7E" is written to the UF0IM1 register to mask all interrupt sources other than CPUDEC interrupt from the interrupt sources indicated by the UF0IS1 register.

(e) "0xF3" is written to the UF0IM2 register to mask all interrupt sources indicated by the UF0IS2 register.

(f) "0xFE" is written to the UF0IM3 register to mask all interrupt sources indicated by the UF0IS3 register.

(g) "0xFF" is written to the UF0IM4 register to mask all interrupt sources indicated by the UF0IS4 register.

(h) "0x03" is written to the KRM register of CPU to detect key interrupt signal.

(i) "0" is written to the USBIF bit of CPU to clear INTUSB interrupt.

(j) "0" is written to the RSUMIF bit of CPU to clear INTRSUM interrupt.

(k) "0" is written to the KRIF bit of CPU to clear INTKR interrupt.

(l) "0" is written to the USBMK bit of CPU to release mask of INTUSB interrupt.

(m) "0" is written to the RSUMMK bit of CPU to release mask of INTRSUM interrupt.

(n) "0" is written to the KRMK bit of CPU to release mask of INTKR interrupt.

(9) Initialization of driver internal flag

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

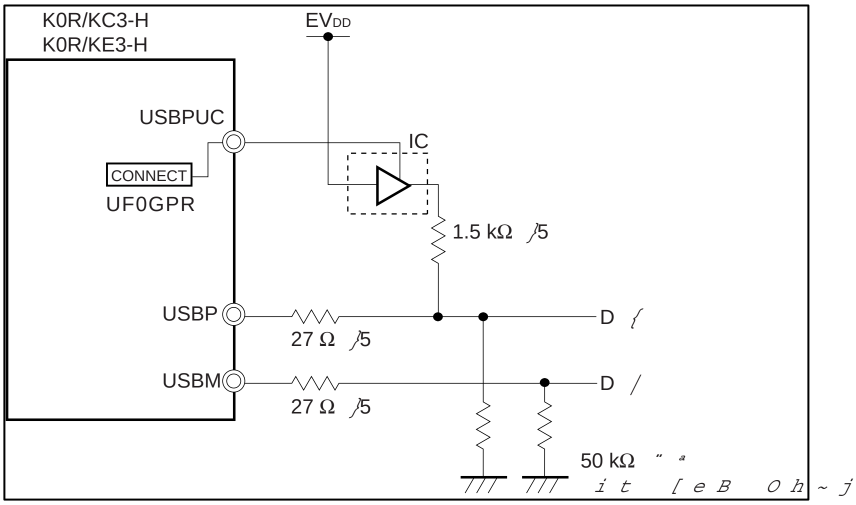
(10) USB buffer enabled/ floating measures disabled

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

(11) Pulling up the D+ signal

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

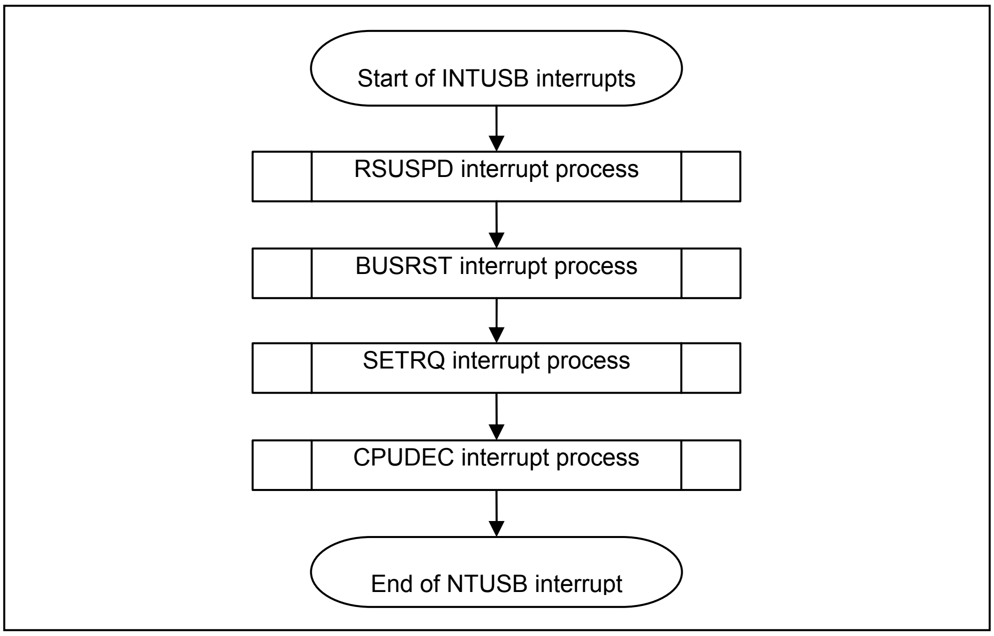
Figure 3-4 USB function controller Connection Example



3.2.3 INTUSB interrupt process

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

Figure 3-5 Process flow of Endpoint0 monitoring



(1) RSUSPD interrupt process

It is reported when Suspend/Resume interrupt is generated. Process is executed in the following order.

(a) This interrupt is generated by Suspend interrupt (verifying that UF0EPS1 is"0x00") and if usbf78k0r_usbstate_flg flag value is "0x04" (Configured state), "0x06" (Suspend state) is written to usbf78k0r_usbstate_flg flag.

(b) usbf78k0r_stop_mode() function is called.

(2) BUSRST interrupt process

It is reported when Bus Reset is generated. Process is executed in the following order.

(a) “0x7F” is written to the UF0IC0 to clear BUSRST interrupt.

(b) “0x05(Bus Reset Occur)” is written to usbf78k0r_usbstate_flg flag.

(3) SETRQ interrupt process

SET_XXXX request for auto process is received and it is reported at auto processing. Process is executed in the following order.

(a) “0xFB” is written to the UF0IC0 to clear SETRQ interrupt.

(b) Both SETCON bit of UF0SET register and CONF bit of UF0MODS register are set to “1” is verified. “1” is set to CONFIGURATION by the SET_CONFIGURATION request is indicated.

(c) “0x04” is written to the usbf78k0r_usbstate_flg flag to report that it is switched from reset state to normal state.

(4) CPUDEC interrupt process

(a) “0xFD” is written to UF0IC1 register to clear PROT interrupt.

(b) UF0E0ST register is read for 8 times then request data is acquired and decoded.

(c) If request is class request, usbf78k0r_classreq() function is called and class request process is executed.

(d) If request is not class request, usbf78k0r_standardreq() function is called and standard request process is executed.

3.3 Function Specification

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 3-12 Functions in the Sample Driver

Source File	Function Name	Description
main.c	cpu_init	Initializes the CPU.
	main	Main routine
usbf78k0r.c	usbf78k0r_init	Initializes the USB function controller
	usbf78k0r_intusbf0	Processing INTUSB interrupt
	usbf78k0r_intkr	Processing INTKR interrupt
	usbf78k0r_intrsum	Processing INTRSUM
	usbf78k0r_stop_mode	Processing STOP mode
	usbf78k0r_standardreq	Processes standard requests
	usbf78k0r_getdesc	Processes GET_DESCRIPTOR(String, HID, Report)
	usbf78k0r_send_EP0	Transmits Endpoint0
	usbf78k0r_receive_EP0	Receives Endpoint0
	usbf78k0r_sendnullEP0	Transmits a NULL packet for endpoint 0.
	usbf78k0r_sendstallEP0	Transmits a STALL for endpoint 0.
	usbf78k0r_ep_status	Notifies FIFO status of bulk/interrupt Inn end point
	usbf78k0r_send_null	Transmits a NULL packet of bulk/interrupt inn endpoint
	usbf78k0r_data_send	Transmits bulk/interrupt Inn end point
usbf78k0r_hid.c usbf78k0r.c	usbf78k0r_classreq	Processing HID class request
	usbf78k0r_get_report	Processing Get Report request
	usbf78k0r_get_idle	Processing Get Idle request
	usbf78k0r_get_protocol	Processing Get Protocol request
	usbf78k0r_set _report	Processing Set Report request
	usbf78k0r_set_idle	Processing Set Idle request
	usbf78k0r_set_protocol	Processing Set Protocol request
	usbf78k0r_init	Initializes the USB function controller
	usbf78k0r_intusbf0	Processing INTUSB interrupt
	usbf78k0r_intkr	Processing INTKR interrupt

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 3-6 Calling Functions in the Main Routine

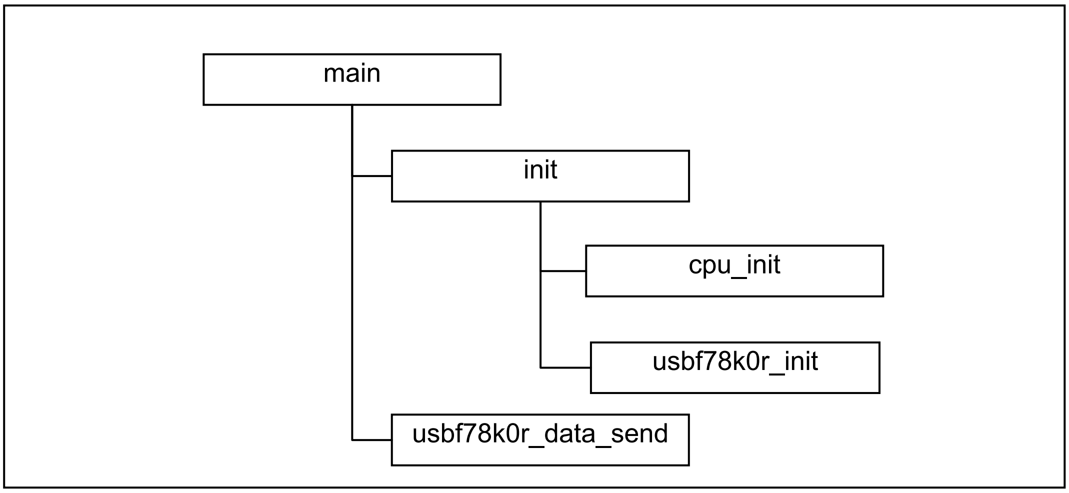


Figure 3-7 Calling Functions during the Processing for the USB function controller

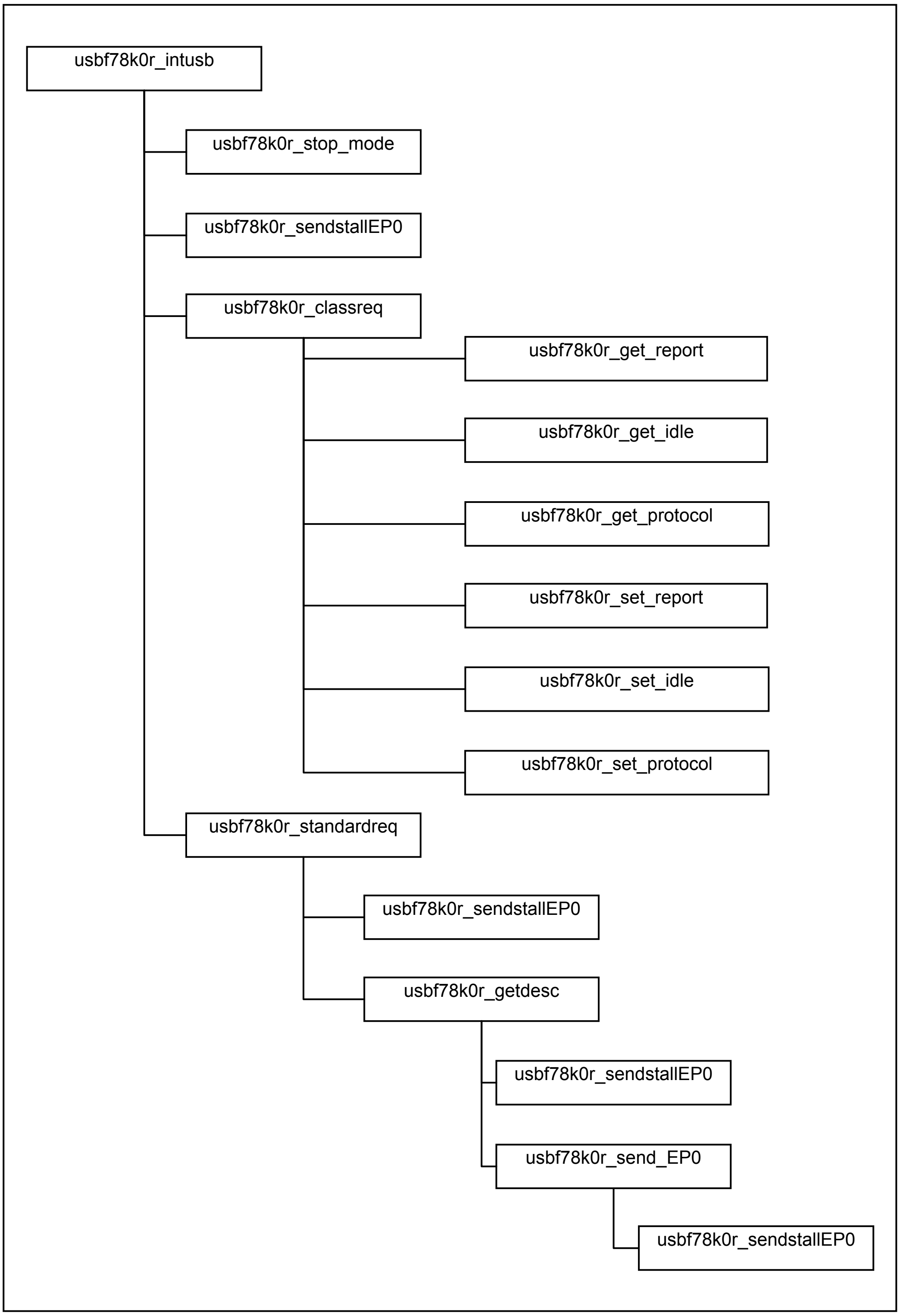
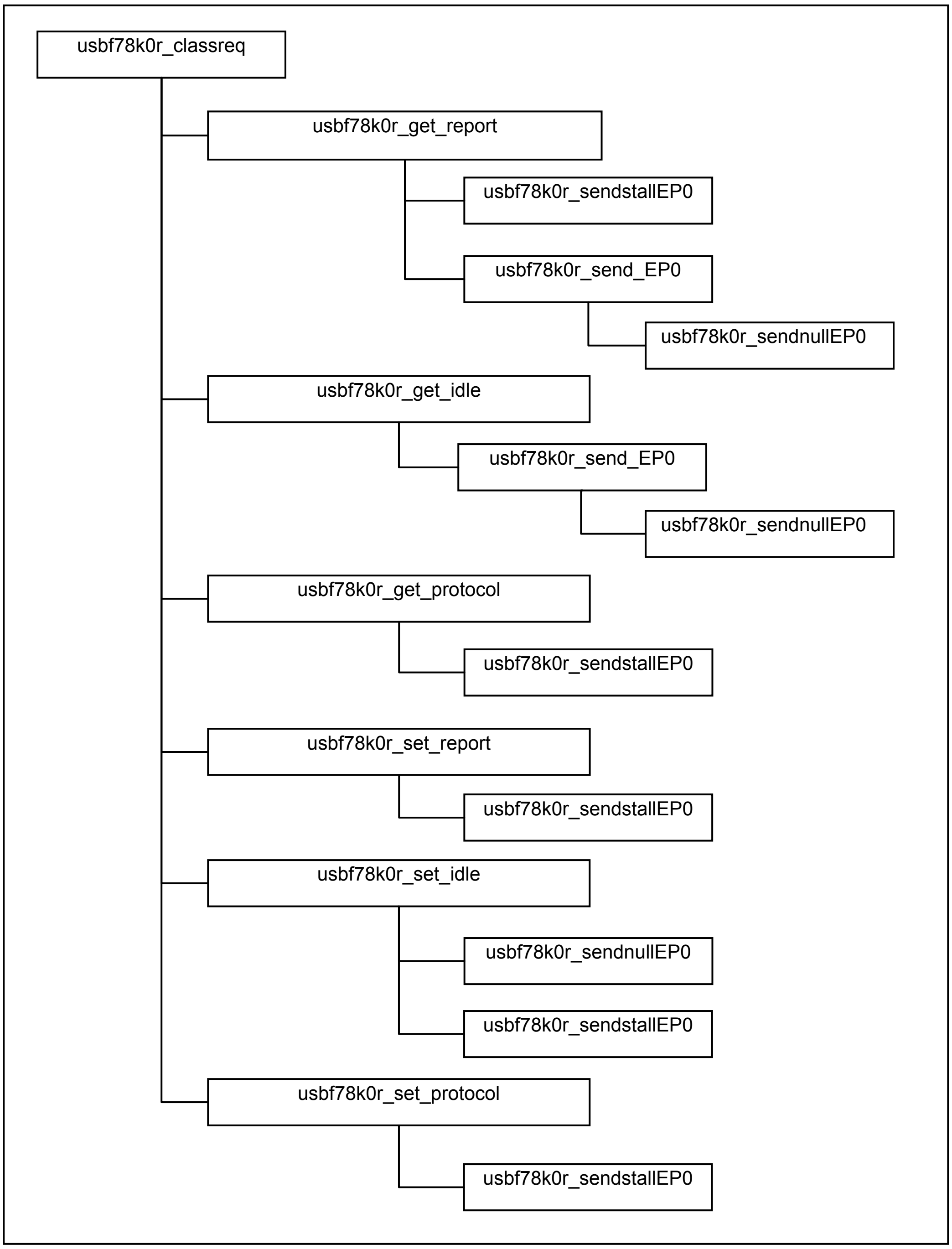


Figure 3-8 Calling Functions during the Processing for the USB HID Class (1)



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 value]
None

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

cpu_init

[Overview]
Initializes the CPU.

[C description format]
void cpu_init(void)

[Parameters]
None

[Return value]
None

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

(3) Functions for the USB function controller

usbf78k0r_init

[Overview]
Initializes the USB function controller

[C description format]
void usbf78k0r_init(void)

[Parameters]
None

[Return value]
None

[Description]
This function is called during initialization processing.
This function specifies the settings required for using the USBF, such as allocating and specifying the data area and masking interrupt requests.

usbf78k0r_intusbf0

[Overview]
INTUSB interrupt processing

[C description format]
__interrupt void usbf78k0r_intusbf0 (void)

[Parameters]
None

[Return value]
None

[Description]
This function is an interrupt service routine called from INTUSBF0 interrupt.
Generated interrupt processing is done while verifying about the interrupt requests about the interrupt which are not masked of USB function controller.

usbf78k0r_intkr

[Overview]
INTKR interrupt processing

[C description format]
__interrupt void usbf78k0r_intkr (void)

[Parameters]
None

[Return value]
None

[Description]
This function is an interrupt service routine called by INTKR interrupt.
Flag indicating "pressed key" is updated.

usbf78k0r_intrsum

[Overview]
INTRSUM interrupt processing

[C description format]
__interrupt void usbf78k0r_intrsum(void)

[Parameters]
None

[Return value]
None

[Description]
This is an interrupt service routine called by INTRSUM interrupt.
Flag indicating "resume signal generation" is updated.

usbf78k0r_stop_mode

[Overview]

Processing to enter in Stop mode at the time of Suspend

[C description format]

void usbf78k0r_stop_mode (void)

[Parameters]

None

[Return value]

None

[Description]

This function is called from the CPUDEC interrupt cause process of INTUSB interrupt process.
It enters in STOP mode by stopping USB clock. STOP mode is cancelled by INTRSUM interrupt, INTKR interrupt.

usbf78k0r_standardreq

[Overview]

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

[C description format]

void usbf78k0r_standardreq (USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

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

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

usbf78k0r_getdesc

[Overview]
Processes GET_DESCRIPTOR requests

[C description format]
void usbf78k0r_getdesc (USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]
None

[Description]
This function is called during the processing of standard requests to which the USB function controller does not automatically respond. If a decoded request requests a string descriptor, this function calls the USB data transmission function (usbf78k0r_send_EP0) for Endpoin0 and transmits a string descriptor from endpoint 0. If a decoded request requests any other descriptor, this function calls the function for returning STALL (usbf78k0r_sendstallEP0) for endpoint 0.

usbf78k0r_send_EP0

[Overview]
Transmits USB data for Endpoint0

[C description format]
INT32 usbf78k0r_send_EP0(UINT8* data, INT32 len)

[Parameters]

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

[Return value]

Symbol	Description
DEV_OK	Normal completion
DEV_ERROR	Abnormal termination

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

usbf78k0r_receive_EP0

[Overview]

Receives USB data for Endpoint0

[C description format]

INT32 usbf78k0r_receive_EP0(UINT8* data, INT32 len)

[Parameters]

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

[Return value]

Symbol	Description
DEV_OK	Normal completion
DEV_ERROR	Abnormal termination

[Description]

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

usbf78k0r_sendnullEP0

[Overview]

Transmits a NULL packet for endpoint 0

[C description format]

void usbf78k0r_sendnullEP0(void)

[Parameters]

None

[Return value]

None

[Description]

This function clears the FIFO for endpoint 0 and transmits a NULL packet from the USBF by setting the bit that indicates the end of data to 1.

usbf78k0r_sendstallEP0

[Overview]
Returns a STALL for endpoint 0

[C description format]
void usbf78k0r_sendstallEP0(void)

[Parameters]
None

[Return value]
None

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

usbf78k0r_ep_status

[Overview]
Notifies FIFO status for bulk/interrupt inn endpoint

[C description format]
INT32 usbf78k0r_ep_status(INT8 ep)

[Parameters]

Parameter	Description
INT8 ep	Data transmission endpoint number

[Return value]

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

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

usbf78k0r_send_null

[Overview]

Transmits a NULL packet for bulk/interrupt in endpoint

[C description format]

INT32 usbf78k0r_send_null(INT8 ep)

[Parameters]

Parameter	Description
INT8 ep	Data transmission end point number

[Return value]

Symbol	Description
DEV_OK	Normal completion
DEV_ERROR	Abnormal termination

[Description]

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

usbf78k0r_data _send

[Overview]

Transmits USB data for bulk/interrupt in endpoint

[C description format]

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

[Parameters]

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

[Return value]

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

[Description]

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

(4) Functions for USB Human Interface Class processing

usbf78k0r_classreq

[Overview]

Processes class request

[C description format]

void usbf78k0r_classreq(USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called from the CPUDEC interrupt cause process of INTUSB interrupt process.
If a decoded request is a communication class request, this function calls each request processing function. For other requests, this function calls the function for returning STALL for Endpoint0 (usbf78k0r_sendstallEP0).

usbf78k0r_get_report

[Overview]

Processes Get Report request

[C description format]

void usbf78k0r_get_report (USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is a Get Report request.
This function transmits current key code from Endpoint0 only when request with report ID 0 is received. Else it responds STALL.

usbf78k0r_get_idle

[Overview]

Processes Get Idle request

[C description format]

void usbf78k0r_get_idle(USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is Get Idle. This function transmits idle rate from Endpoint0. Idle rate of sample driver is fixed to 0 (transmitted only when it changes).

usbf78k0r_get_protocol

[Overview]

Processes Get Protocol request

[C description format]

void usbf78k0r_get_protocol(USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is Get Protocol. It responds STALL.

usbf78k0r_set_report

[Overview]

Processes Set Report request

[C description format]

void usbf78k0r_set_report (USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is Set Report. It responds STALL.

usbf78k0r_set_idle

[Overview]

Processes Set Idle request

[C description format]

void usbf78k0r_set_idle(USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is Set Report. It transmits Null packet when idle rate transmitted by host is 0, else it responds STALL.

usbf78k0r_set_protocol

[Overview]

Processes Set Protocol request

[C description format]

void usbf78k0r_set_protocol(USB_SETUP *req_data)

[Parameters]

Parameter	Description
USB_SETUP *req_data	Request data storage pointer address

[Return value]

None

[Description]

This function is called if request decoded at class request process is Set Protocol. It responds STALL.

Chapter 4 Sample Application Specification

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

4.1 Overview

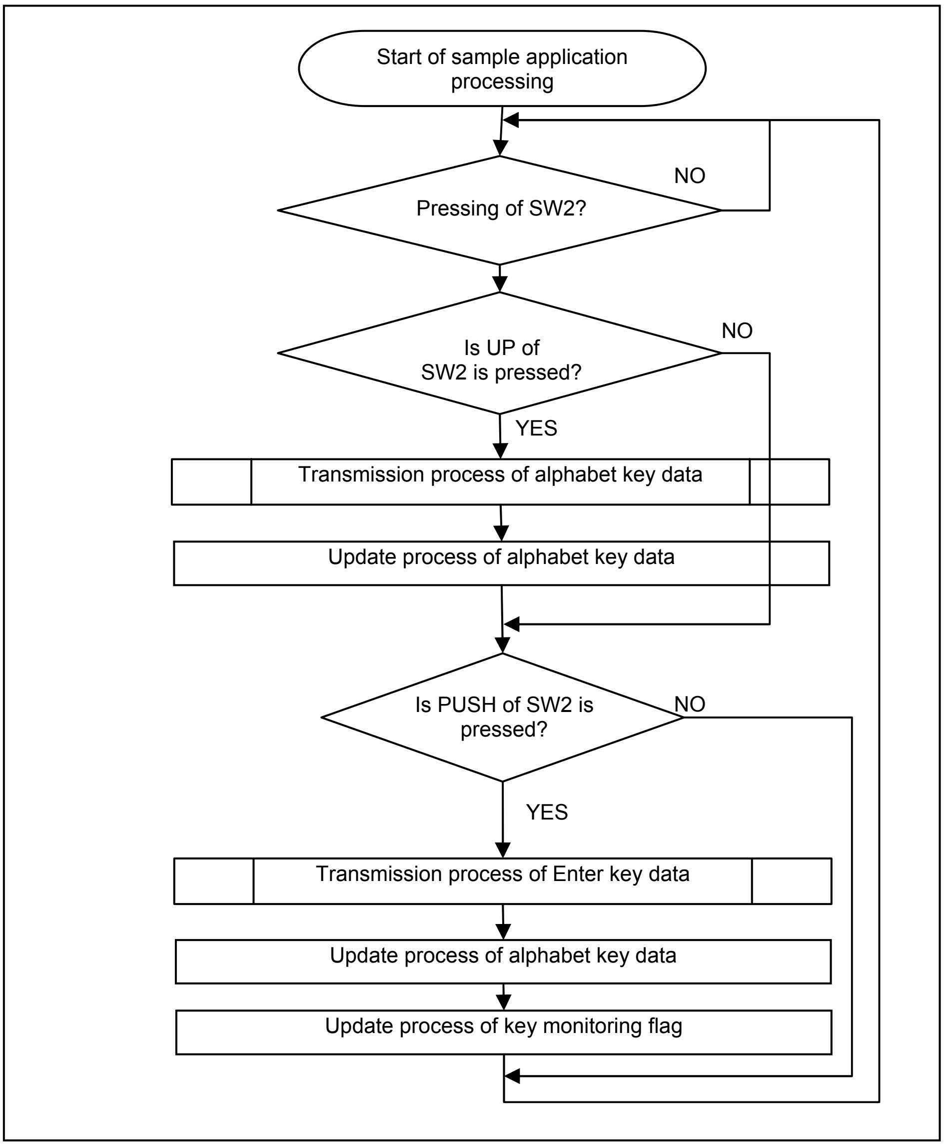
The sample application is provided as a simple example of using the USB human interface device class driver and is incorporated in the main routine of the sample driver.

This sample applications transmits alphabets “a” ~ “z” when switch SW2(PUSH) is pressed on the TK-78K0R/KE3L+USB board and transmits “Enter” at pressing the SW2(UP). Various functions of the sample driver are used during this processing.

4.2 Operation

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

Figure 4-1 Flowchart for the Sample Application Processing



(1) Verifying whether SW2 is pressed

Whether switch SW2 UP or PUSH is pressed in the TK-78K0R/KE3L+USB board is verified from the key pressing verification flag (usbf78k0r_keytouch_flg). If flag value is “1”, key data is transmitted but if flag value is “0”, no transmission.

(2) Verifying whether SW2 UP is pressed

If P70 is “0”, it is determined that SW2 UP is pressed and wait till gets off.

(3) Transmission process of alphabet key data

It is verified that FIFO of endpoint for transmission is Null by usbf78k0r_ep_status function and then alphabet key data and release key data are transmitted by usbf78k0r_data_send function.

(4) Alphabet key data information update

It updates alphabet key data (Toggle). If key data is “z”, it is “a”.

(5) Verifying whether SW2 PUSH is pressed

If P71 is “0”, it is determined that SW2 PUSH is pressed and waits till gets off.

(6) Transmission process of Enter key data

It is verified that FIFO of endpoint for transmission is Null by usbf78k0r_ep_status function and then enter key data and release key data are transmitted by usbf78k0r_data_send function.

(7) Alphabet key data information update

It updates alphabet key data (return to “a”).

(8) Update flag for verification of pressing of key

It updates (set to “0x00”) (usbf78k0r_keytouch_flg) flag for verification of pressing of key.

4.3 Using functions

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

(1) Definitions and declarations

“usbf78k0r.h” header file is included in order to use the sample driver functions. Array (keycode) for key data storage is set.

(2) Initialization process of CPU

Initialization process of CPU function (cpu_init) is called.

(3) Initialization process of USB function controller

USB function controller initialization function (usbf78k0r_init) is called.

(4) Verification of FIFO status for user data

FIFO status is verified by FIFO status verification function (usbf78k0r_ep_status).

(5) Transmitting user data

Data is transmitted by data transmission function (usbf78k0r_data_send) after specifying data, data size, and endpoint.

List 4-1 Sample Application Code (Portion)

```
void main( void )
{
   UINT8 keycode[REPORT_DATA_LENGTH];
   UINT8 keydata = A_KEY;

   init();

   memset(keycode, 0, sizeof(keycode));  /* Key Data Clear */
   usbf78k0r_keytouch_flg = F_SW_OFF;      /* Key Flag Clear */

   while(1)
   {
       if ( usbf78k0r_keytouch_flg == F_SW_ON ) {
          if((P7 & 0x01) == 0x00) {/* SW2 UP */
             while((P7 & 0x01) == 0x00){
             * SW2 OFF WAIT */
             }
             keycode[KEY1_SCAN_CODE] = keydata; /* Press Key Data */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
             memset(keycode, 0, sizeof(keycode)); /* Release Key Data */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
             keydata++;                         /* a to z */
             if(keydata == EXCLAMATION_KEY) {
             keydata = A_KEY;
             }
          }
          if((P7 & 0x02) == 0x00) {/* SW2 PUSH */
             while((P7 & 0x02) == 0x00){
             /* SW2 OFF WAIT */
             }
             keycode[KEY1_SCAN_CODE] =ENTER_KEY;/*Press Key Data(Enter) */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
             memset(keycode, 0, sizeof(keycode)); /* Release Key Data */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
             keydata = 0x04;
          }
          usbf78k0r_keytouch_flg = F_SW_OFF;
       }
   }
}
```

Chapter 5 Development Environment

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

5.1 Development environment overview

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 machine	-	PC/AT compatible computer (OS : Windows XP)
Software	Integrated development tool	IAR Embedded Workbench for 78K	V4.70
	Compiler	ICC78K0R	V4.70.1
	Assembler	A78K0R	V4.70.1

5.1.2 Debugging

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

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

Components		Product Example	Remark
Hardware	Host machine	-	PC/AT compatible computer (OS : Windows XP)
	Target device	TK-78K0R/KE3L+USB	
	In circuit emulator	MINICUBE2	
	USB cables	-	miniB-to-A connector cable
Software	Integrated development tool	IAR Embedded Workbench for 78K	V4.70
	Debugger	IAR C-SPY debugger	V4.70.1

5.2 Setting up the Environment

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

5.2.1 Preparing Host Environment

Open the dedicated workspace on the host for debugging the sample application.

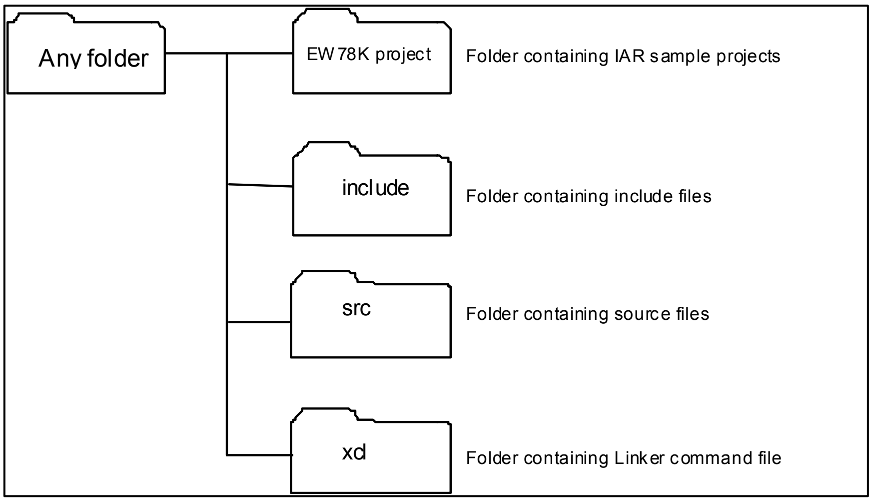
(1) Installing the Integrated development tool

Install the IAR Embedded Workbench for 78K. For details, see the **IAR Embedded Workbench for 78K User's Manual**.

(2) Copying drivers

Store the set of files provided with the sample driver in any directory without changing the folder structure. You can store it in any directory on your host system hard drive.

Figure 5-1 Folder Structure of the Sample Driver

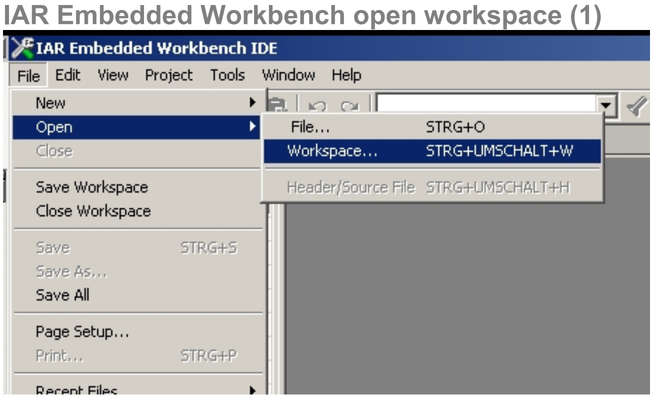


(3) Loading the HID driver Workspace

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

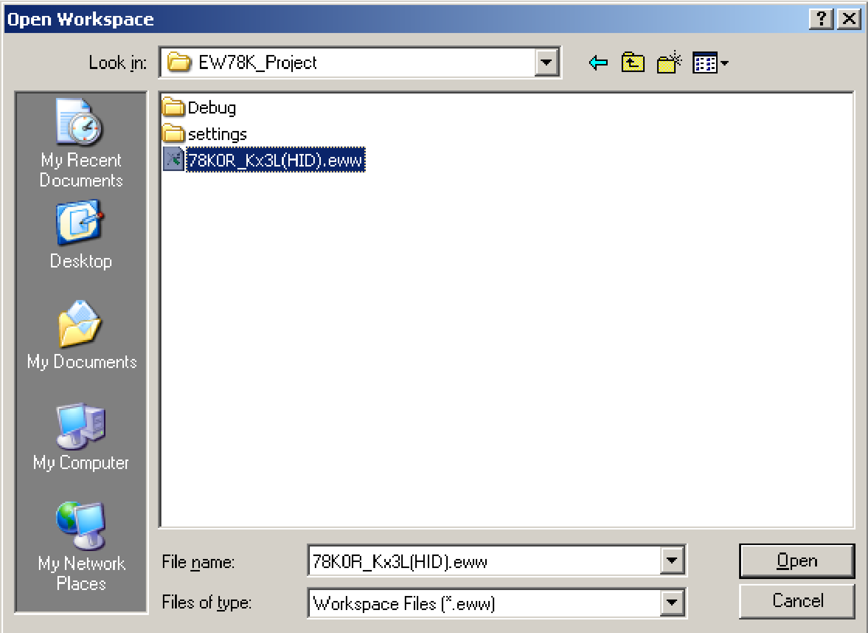
(a) Start the IAR Embedded Workbench for 78K, and then select "**Open → Workspace"** in the "**File"** menu.

Figure 5-2 IAR Embedded Workbench open workspace (1)



(b) In the **Open Workspace** dialog box, specify the workspace file (78K0R_Kx3L(HID).eww) in the EW78K_project folder, which is the sample driver installation directory.

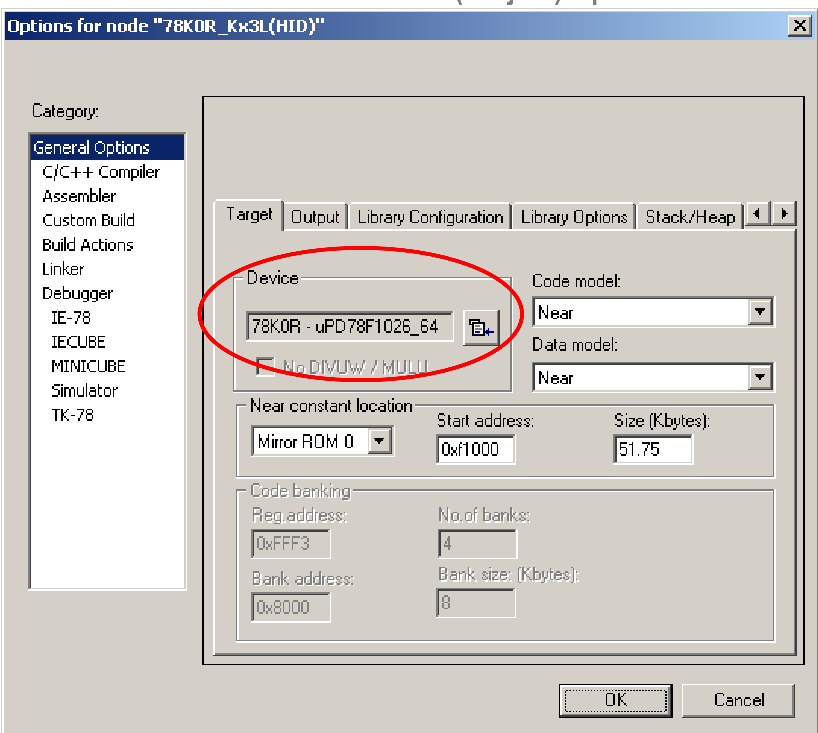
Figure 5-3 IAR Embedded Workbench open workspace (2)



(4) Verify that the correct device is selected

To make sure that the correct device is selected in this project open the Project options by clicking "**Project**" → "**Options**" and check that the "78K0R – uPD78F1026_64" is chosen as **Device**.

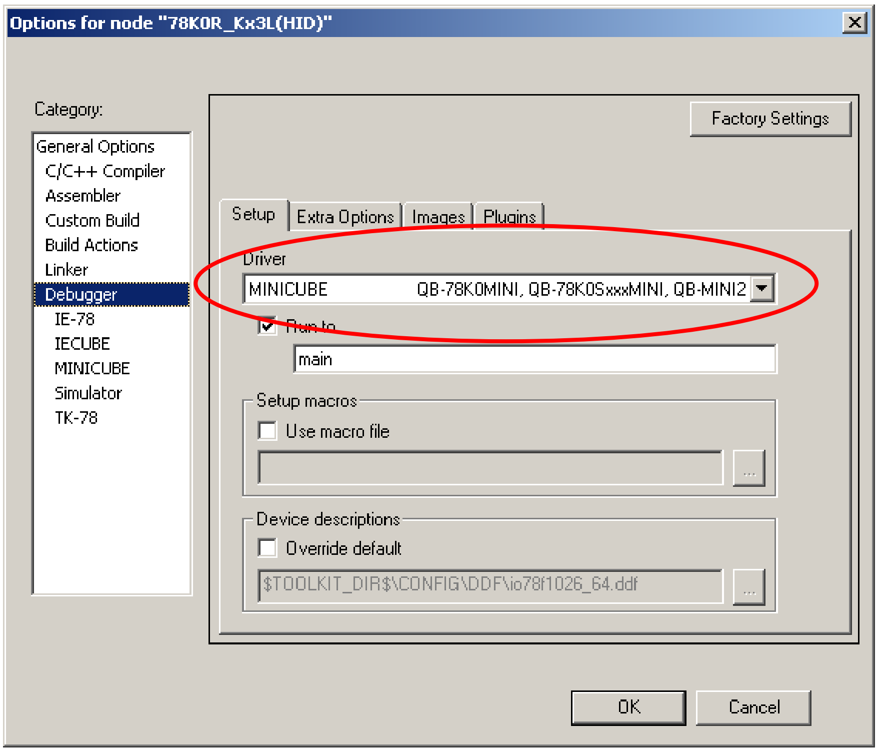
Figure 5-4 IAR Embedded Workbench General (Project) Options



(5) Verify that the correct debugger is selected

To make sure that the correct Debugger is selected, switch to the Debugger menu in the Project Options and verify that **MINICUBE** is selected as **Driver**.

Figure 5-5 IAR Embedded Workbench Debugger Options



Note Do not close the IAR Embedded Workbench for 78K now, you will need it later.

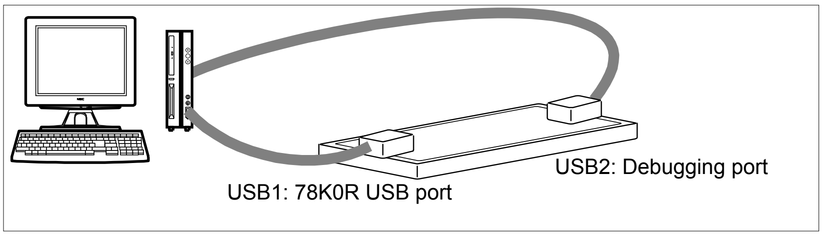
5.2.2 Setting up the target environment

Connect the target device to use for debugging.

(1) Connecting the target device

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

Figure 5-6 Connecting the TK-78K0R/KE3L+USB



(2) Installing the host driver

The procedure for using the virtual COM port host driver included with the sample driver is described in the starter kit User's manual (R20UT0010ED0100_78k0rkx3l.pdf) chapter **USB Driver installation**. This document is also available on the Starter Kit CD-ROM.

5.3 On-Chip Debugging

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

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

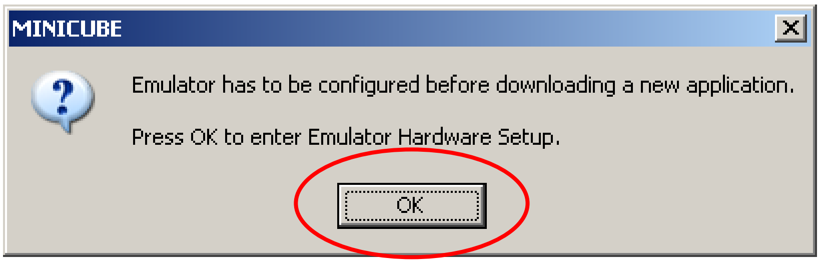
5.3.1 Generating the debug files

To write a program to the target device, you need to generate a machine code file including debug information from the given HID sample project. To do so return to the IAR Embedded Workbench for 78K and generate the output files by clicking "**Project**" → "**Make**" or pressing the **Make** button ( ).

5.3.2 Download and Debug

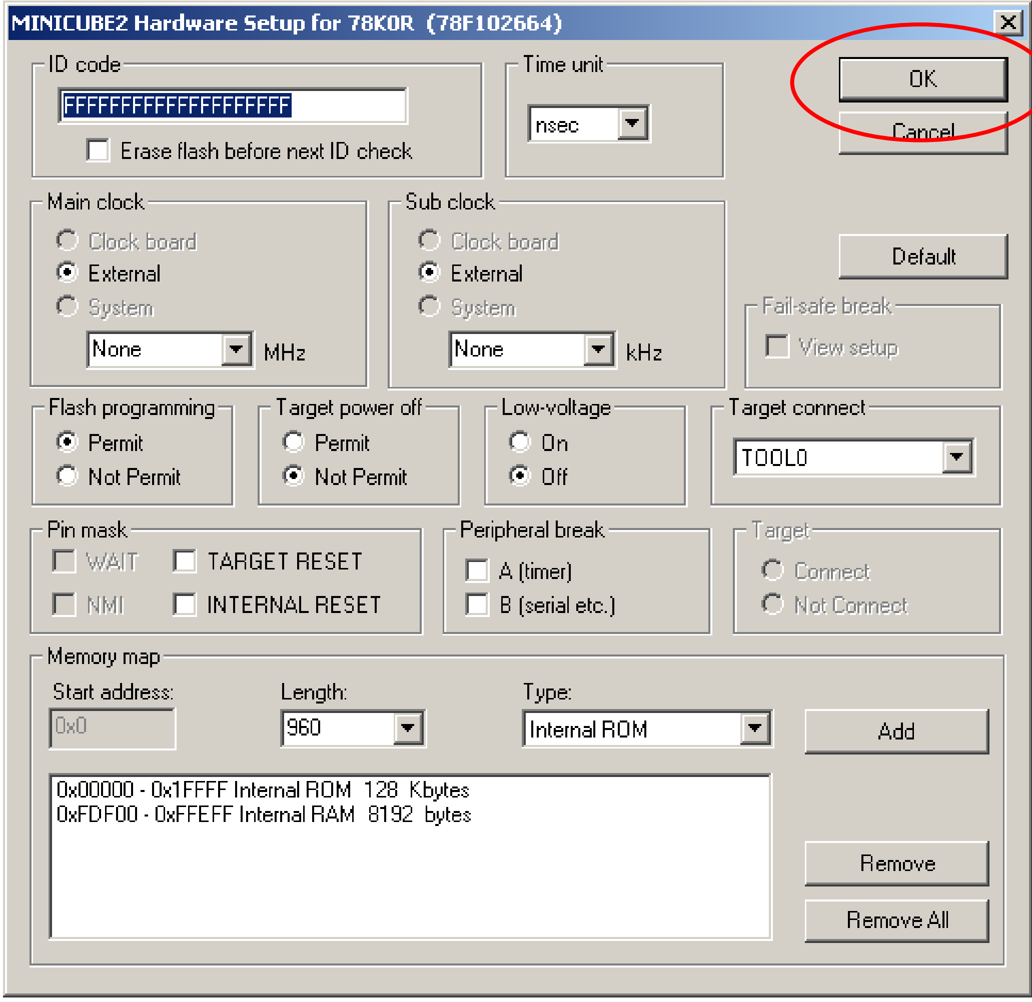
After the output files are generated, they can be downloaded to the target device, using the IAR C-SPY debugger. To do so just click on "**Project**" → "**Download and Debug**" or use the **Download and Debug** button ( ). When starting the first debug session the communication interface has to be configured. The following message will occur. Press OK to open to the configuration window.

Figure 5-7 IAR C-SPY debug interface configuration (1)



The Hardware setup window will occur. As the default hardware configuration can be used for this all settings can be left untouched and only the OK button has to be pressed.

Figure 5-8 IAR C-SPY debug interface configuration (2)

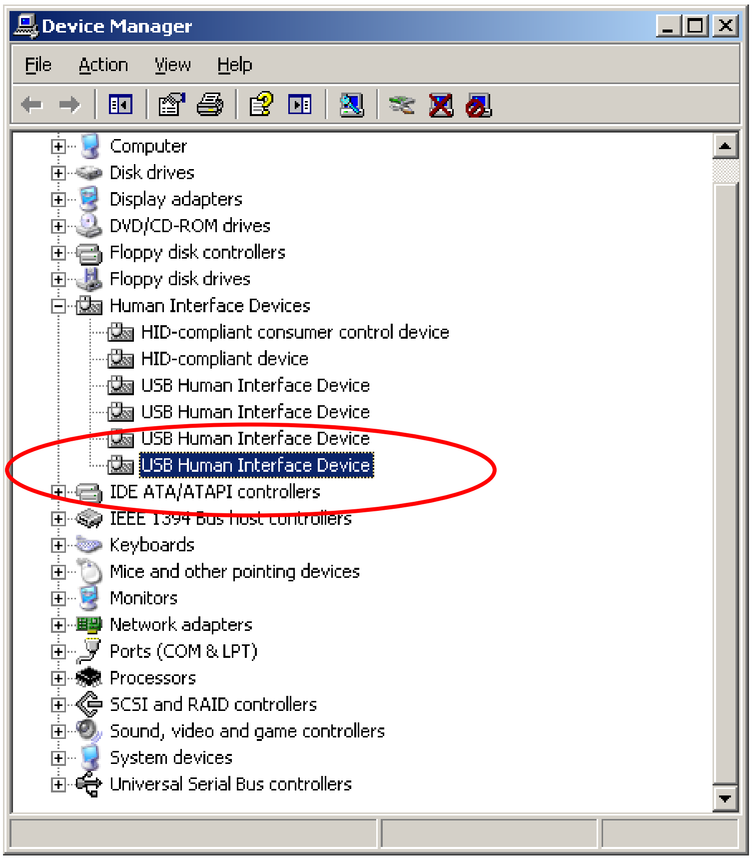


When the download of the program is finished the IAR C-SPY debugger window will open up, the HID sample project will run to the beginning of the **main** function and will break at this point.

To start the application, click “**Debug**” → “**Go**” or press the **Go** button (). When running the HID sample application the first time the Windows new Hardware detection will recognize the device and the windows driver for a standard HID device will be installed automatically.

To check that the device driver is installed correctly open the Windows Device Manager window. In the Human Interface Devices category, you will find an additional **USB Human Interface Device**.

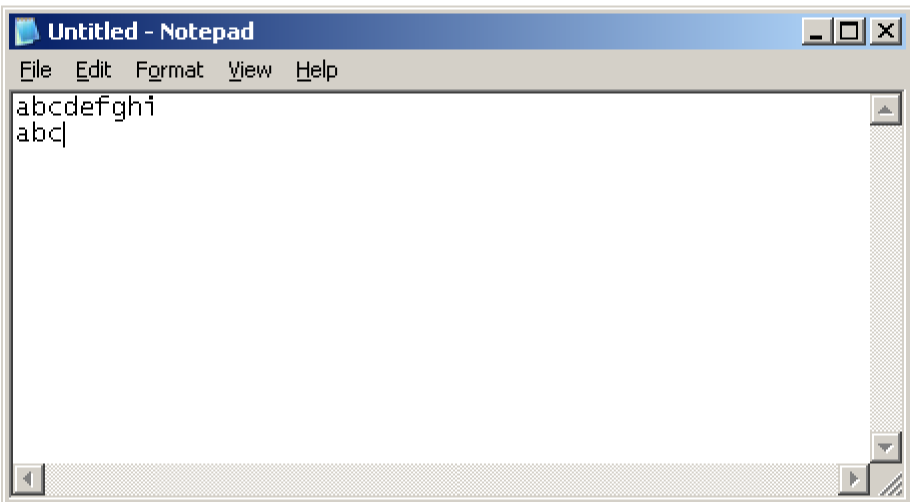
Figure 5-9 Windows Device Manager



5.4 Checking the Operation

If the HID sample driver is running in the IAR C-SPY debugger and the standard HID driver is installed in Windows you can check its operation by opening an editor software (such as Notepad) on the host PC. The device transmits (“a” to “z”) at sequential toggling of “Enter” key in SW2 PUSH and alphabets “a” ~”z” in SW2 UP on the TK-78K0R/KE3L+USB.

Figure 5-10 Notepad output of HID input



Chapter 6 Using the Sample Driver

This chapter describes information that you should know when using the USB human interface class sample driver for the 78K0R/Kx3-L.

6.1 Overview

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

(1) Customizing the sample driver

Rewrite the following sections of the sample driver as required.

- The sample application section in “main.c”
- The values specified for the various registers in “usbf78k0r.h” file
- The descriptor information in “usbhid_desc.h” file

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

(2) Using functions

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

6.2 Customizing the sample driver

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

6.2.1 Application section

The code in main.c file below includes a simple example of processing the sample driver. Rewrite this part to realize your personalized application.

List 6-1 Sample Application Code

```
void main( void )
{
   UINT8 keycode[REPORT_DATA_LENGTH];
   UINT8 keydata = A_KEY;

   init();

   memset(keycode, 0, sizeof(keycode)); /* Key Data Clear */
   usbf78k0r_keytouch_flg = F_SW_OFF;      /* Key Flag Clear */

   while(1)
   {
       if ( usbf78k0r_keytouch_flg == F_SW_ON ) {
          if((P7 & 0x01) == 0x00) {/* SW2 UP */
             while((P7 & 0x01) == 0x00){
             * SW2 OFF WAIT */
             }
             keycode[KEY1_SCAN_CODE] = keydata; /* Press Key Data */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
             memset(keycode, 0, sizeof(keycode)); /* Release Key Data */
             while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
             usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
```

```
                keydata++;                              /* a to z */
                if(keydata == EXCLAMATION_KEY) {
                keydata = A_KEY;
                }
            }
            if((P7 & 0x02) == 0x00) {/* SW2 PUSH */
                while((P7 & 0x02) == 0x00){
                /* SW2 OFF WAIT */
                }
                keycode[KEY1_SCAN_CODE] =ENTER_KEY;/*Press Key Data(Enter) */
                while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
                usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
                memset(keycode, 0, sizeof(keycode)); /* Release Key Data */
                while (usbf78k0r_ep_status(C_INT1) != DEV_OK) {}
                usbf78k0r_data_send(keycode, sizeof(keycode), C_INT1);
                keydata = 0x04;
            }
            usbf78k0r_keytouch_flg = F_SW_OFF;
        }
    }
}
```

6.2.2 Setting up the device registers

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

6.2.3 Descriptor information

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

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

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.

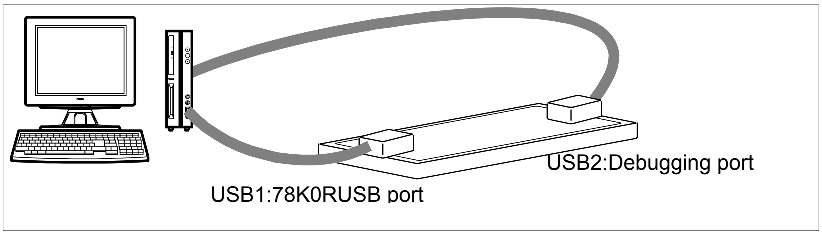
Chapter 7 Starter Kit

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

7.1 Overview

TK-78K0R/KE3L+USB is a kit to develop applications that use the 78K0R/KE3-L microcontroller. 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 7-1 Connections of TK-78K0R/KE3L+USB



7.1.1 Features

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

- A USB miniB connector for the internal USBF
- As small as a business card
- Efficient development by using the board with the integrated development environment (IAR Embedded Workbench for 78K)

7.2 Specification

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

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

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.

[MEMO]

Sales Offices

Renesas Electronics Corporation

www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

R01AN0009ED0100