Renesas USB MCU and USB ASSP

USB Basic Firmware Installation Guide

R01AN0495EJ0200
Rev.2.00
Jun 1, 2012

Introduction

This document is an installation guide for the Renesas USB Basic firmware (herein referred to as USB Basic Firmware Installation Guide), a sample program for Renesas USB MCU and USB ASSP.

This document is intended for use with the application note for the Renesas USB MCU and USB ASSP USB Basic firmware and the device user's manual (Hardware) when developing software.

The USB Basic Firmware Installation Guide (Application Notes and Sample codes) are available for download from the Renesas web.

Renesas Website: http://www.renesas.com/

Product Name:

Renesas USB MCU and USB ASSP USB Basic Firmware

File Name:

an_r01an0512ej_usb_device.zip

Target Device

RX62N Group, RX621 Group, RX630 Group, RX63N Group, RX631Group, R8A66597

The program described here can be used with other RX600 Series microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

This program has been evaluated using Renesas Starter Kit + for RX62N board and Renesas Starter Kit + for RX63N board and Renesas Starter Kit for RX630 board.

Contents

1. Introduction

1.1 Feature Overview

The main features of USB Basic Firmware Installation Guide are as follows.

[Overall]

Can control RX62N, RX63N, RX630 and R8A66597 by common source code.
Can operate in either host or peripheral mode.
Multiple device class drivers may be installed without the need to customize USB Basic Firmware Installation Guide.
USB Basic Firmware Installation Guide does not support multiple interfaces.

[Host mode]

When a no-response condition is detected during data transfer to a USB Function, the user can specify the number of retries per transfer.
A single pipe can perform multiple exclusive data communication tasks.
A common API for control transfer, bulk transfer and interrupt transfer is provided.
The function R_usb_hstd_ChangeDeviceState for devices connect/disconnect processing is provided.
The function R_usb_hstd_ChangeDeviceState for suspend/resume processing is provided.
HUBCD sample program code is provided.
Sample application for data transfer is added. (This application operates as Vendor class.)
A single pipe can perform multiple exclusive data communication tasks in order to manage HDCD pipe information tables.

[Peripheral mode]

Operation can be confirmed by using *USBCommandVerifier.exe*.
(USBCV is available for download from http://www.usb.org/developers/developers/tools/.)
API for control transfer is provided.
Common API for bulk transfer and interrupt transfer is provided.
An API function is provided for devices connect/disconnect processing is provided.
An API function is provided for suspend/resume processing is provided.
Sample application for data transfer is added. (This application operates as Vendor class.)

The following functions must be provided by the customer to match the system under development.

Overcurrent detection processing when connecting USB cables (Host mode).
Descriptor analysis (Host mode).
Descriptor and pipe data (Peripheral mode)
Device class driver.Examples currently exist for HMSC, PMSC, HHID, PHID, HCDC, PCDC.

1.2 Preparation

The following describes the evaluation environment required to operate this firmware, including the list of necessary devices.

Evaluation Board

— RX62N Group/RX621 Group :
Renesas Starter Kit+ for RX62N (RX62N-RSK): Type name:R0K5562N0S000BE
Renesas Starter Kit+ for RX62N: a complete development kit for Renesas RX62N Group/RX621 Group (manufactured by Renesas Electronics)

— RX63N Group :
Renesas Starter Kit for RX630 (RX630-RSK):Type name: R0K50563NC000BR
Renesas Starter Kit + for RX63N: a complete development kit for Renesas RX630 Group (manufactured by Renesas Electronics)

→Refer to the Starter Kit Instruction Manual for setup details.

Development Environment

High-performance Embedded Workshop 4, the integrated development environment (manufactured by Renesas Electronics)
C/C++ Compiler Package for RX Family (manufactured by Renesas Electronics)
E1 Emulator or E20 Emulator (manufactured by Renesas Electronics)
Real-Time OS [RI600/4] for RX Family (supports ulTRON OS) (manufactured by Renesas Electronics)
→Refer to the corresponding instruction manual for details concerning each Renesas development environment.

Other

Emulator Host PC (Windows® 2000, Windows® XP)
USB Host PC (Windows® 2000, Windows® XP, Windows® Vista, Windows® 7) (when developing a USB peripheral system)
Target USB device (when developing a USB host system)
USB cable

2. Operating USB Basic Firmware Installation Guide

2.1 Sample Application Operation Confirmation

2.1.1 Sample Application Operating Specifications

The USB Basic Firmware Installation Guide comes with a sample vendor class driver and application. USB Bulk and Interrupt data transfers can be executed without modification to the firmware.

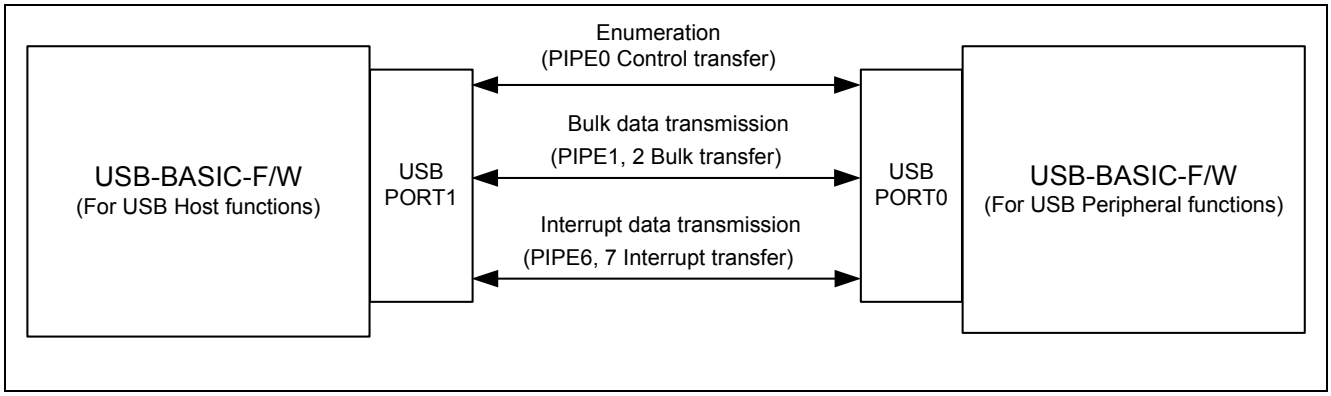

Figure.2.1 USB Basic Firmware Installation Guide USB Data Transfer Image (fot the RX62N and the RX63N)

PIPE configurations are provided below. Note: transfer data is dummy data (00, 01, 02…).

Table2.1 PIPE Configurations

PIPE No.	Transfer Type	Max. Packet Size
PIPE0	Control	64 bytes
PIEP1	Bulk IN	64 bytes
PIPE2	Bulk OUT	64 bytes
PIPE6	Interrupt OUT	64 bytes
PIPE7	Interrupt IN	64 bytes

2.1.2 Setup

(1) Hardware

The connection topology for each device is described below.

Please refer to the corresponding instruction manual concerning evaluation board setup, emulator use, etc.

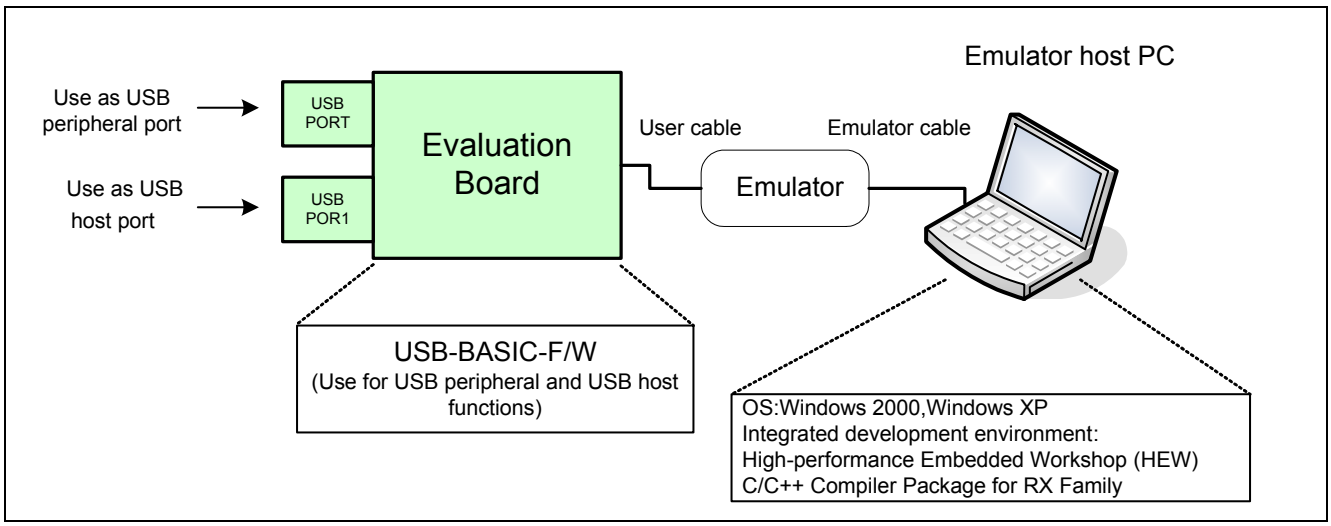

Figure..2.2 Device Connection Topology

The evaluation board of confirmed operation is described below.

Table 2.1 The evaluation board of confirmed operation

MCU	Evaluation Board	USB port(Default action)
RX62N	RX62N-RSK	USB0 (Peripherl mode)
		USB1 (Host mode)
RX630	RX630-RSK	USB0 (Peripherl mode)
RX63N	RX63N-RSK	USB0 (Host mode)
		USB1 (Peripherl mode(static))

(2) Software

Source codes forUSB Basic Firmware Installation Guide can be compiled, linked and decoded in HEW4. To start up HEW4, double click on the HEW workspace file (FW.hws) provided.

For evaluation board , upload the USB Basic Firmware Installation Guide sample program and execute according to the following steps.

Select the build configuration as follows:

Build configuration selection

Select one of the following. Only the selected source files will be compiled.
RX630 can only be set as follows 2.

And before build, USB mode (Host/Peripheral) sets macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h)

1.HOST: Operate one port as host.

2.PERI: Operate one port as peripheral.

3.PERI_HOST: Operate both ports; one as peripheral and one as host.

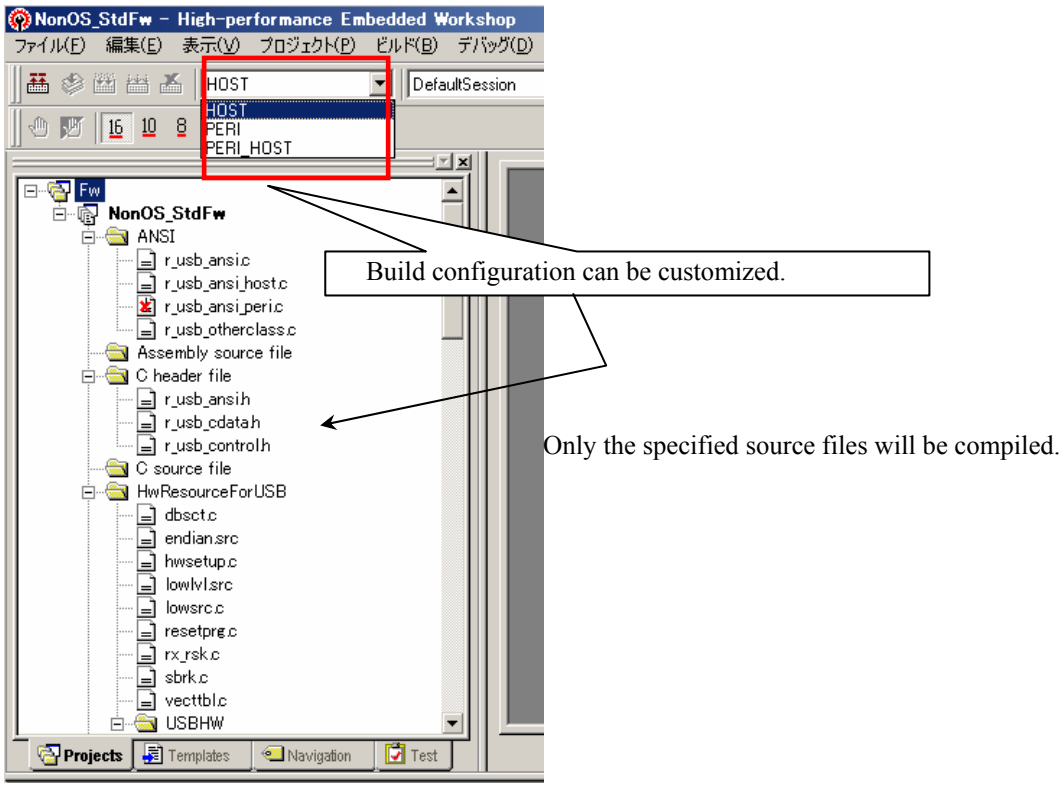

Figure .2.3 Project Selection Examples (using HEW)

1) Select [**Build configuration**] (HOST, PERI or PERI_HOST)
2) You will be prompted to select the emulator mode; select the mode according to the emulator you are using.
3) Download the program by selecting: [**Debug → Download → All Download Modules**].
4) Execute the program by selecting: [**Debug → Execute after reset**].

2.1.3 Sample Application Operation Image

(1) When running the sample application on one evaluation board (for the RX62N-RSK and the RX63N-RSK)

USB Basic Firmware Installation Guide operations can be confirmed by selecting PERI_HOST in the build configuration in the HEW environment, as shown in the connection image below. USB mode (Host/Peripheral) sets macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h)

The USB bus analyzer can be used to monitor the transmission contents of the USB data transfer.

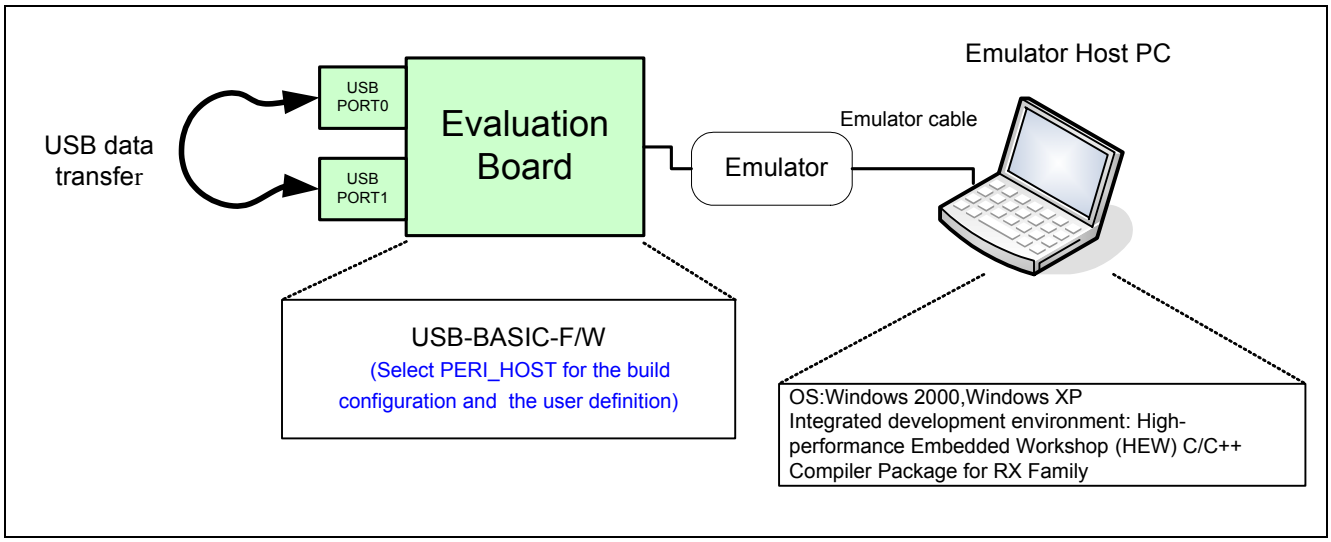

Figure.2.4.Using one evaluation board (for the RX62N-RSK and the RX63N-RSK)

(2) When running the sample application on two evaluation boards

Use one evaluation board for the USB host functions and the other for the USB peripheral functions. Select PERI (peripheral mode) and HOST (host mode), respectively, in the HEW environment build configuration for each USB Basic Firmware Installation Guide, and execute the program. Confirm operations as shown in the connection image below.USB mode (Host/Peripheral) sets macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h)

The USB bus analyzer can be used as the transmission contents of the USB data transfer.

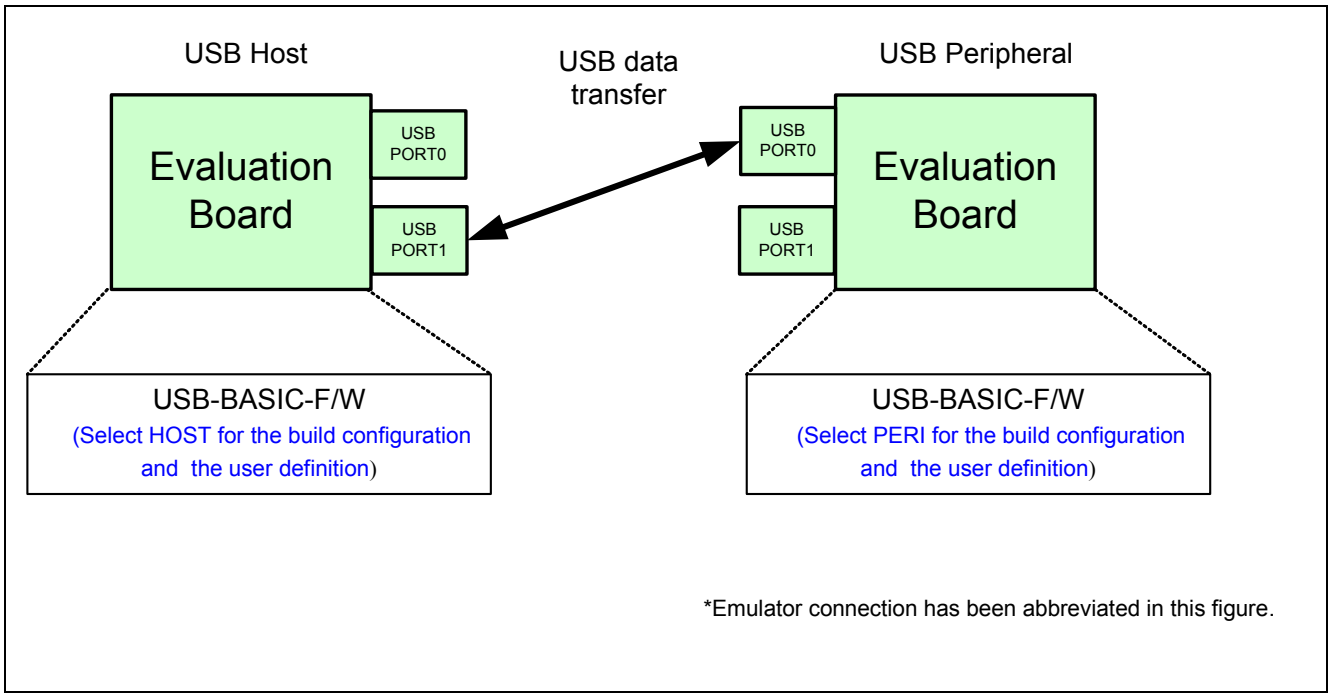

Figure.2.5. Using two evaluation boards

2.2 Confirming Operations with USBCV

When using USB Basic Firmware Installation Guide with USB peripheral functions, operations can be confirmed with the "USBCommandVerifier" (USBCV) Chapter9 test without modifying USB Basic Firmware Installation Guide.USB mode (Host/Peripheral) sets macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h)

The connection topology for each device is shown below.

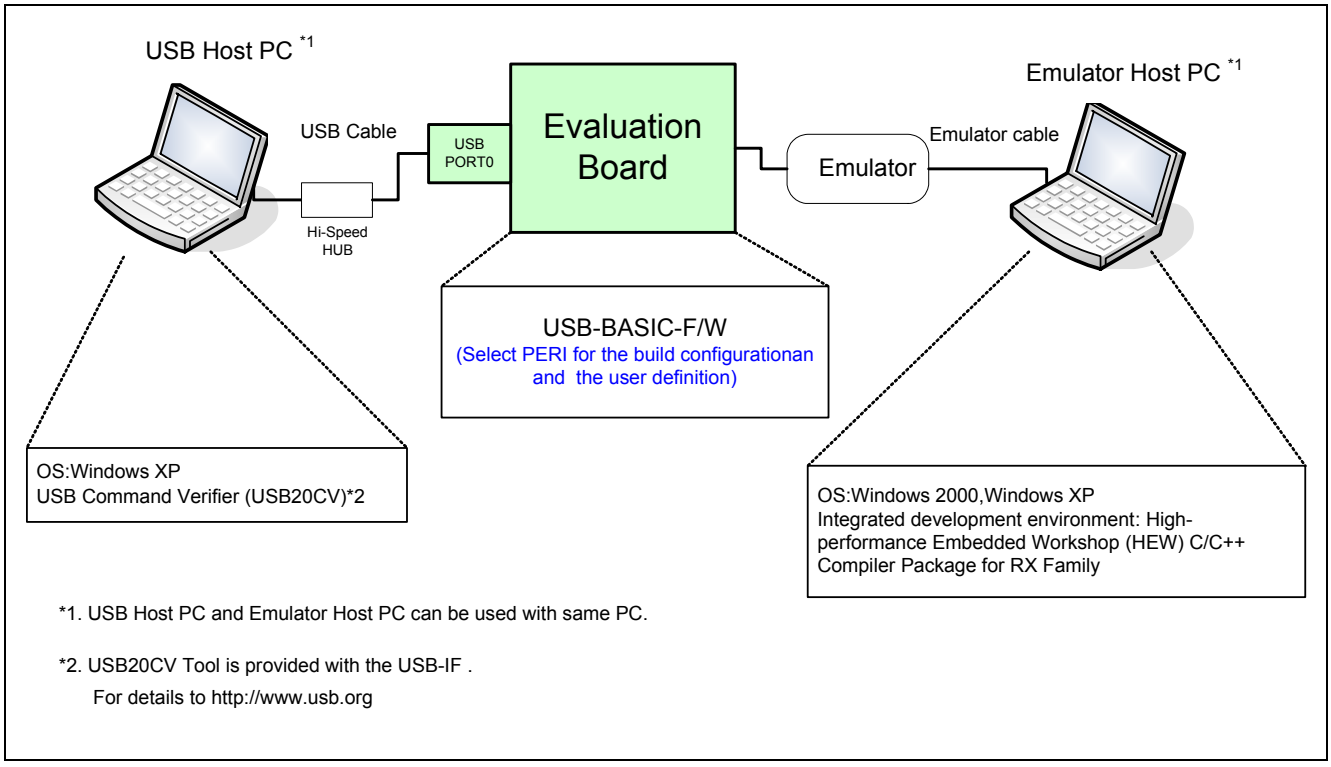

Figure .2.6 Example of Confirming Operations with USBCV (USB peripheral functions)

3. Customized USB Basic Firmware Installation Guide Operations

USB Basic Firmware Installation Guide can operate as a USB driver to match the user system by selecting hardware resours for device (HwResourceForUSB), setting user defined information (“r_usb_usrconfig.h”), and workspace settings..For more details, refer to the Renesas USB MCU and USB ASSP USB Basic Firmware Application Note (Document No. R01AN0512EJ).

3.1 Peripheral mode example

The following uses RX62N in an example to show how to setup the USB Basic Firmware Installation Guide and sample program to operate as a USB peripheral.

1. Select HwResourceForUSB

Delete the content of folder "*HwResourceForUSB*”. Copy and paste the content of "*HwResourceForUSB_RX62N*" into "*HwResourceForUSB*". The folders named *HwResourceForUSB_XXX* are not used by any build configuration.

2. Select USB mode (Host/Peripheral)

Set macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h) as follows.

```
//#define    USB_FUNCSEL_USBIP0_PP    USB_HOST_PP     // Host Mode
#define      USB_FUNCSEL_USBIP0_PP    USB_PERI_PP     // Peripheral Mode
//#define    USB_FUNCSEL_USBIP0_PP    USB_NOUSE_PP

//#define    USB_FUNCSEL_USBIP1_PP    USB_HOST_PP     // Host Mode
//#define    USB_FUNCSEL_USBIP1_PP    USB_PERI_PP     // Peripheral Mode
#define      USB_FUNCSEL_USBIP1_PP    USB_NOUSE_PP
```

3. Workspace

Double click “Fw.hws” to start up HEW. Then set the build configuration in the workspace to “PERI”.

4. Generate an executable file

From the tabs at the top of the workspace, select [Build → Build all], then execute the build.

5. Connect to evaluation board

From the tabs at the top of the workspace, select [Debug→ Connect], and then connect the evaluation board to the emulator.

6. Download and execute the binary executable in the target

From the tabs at the top of the workspace, select [Debug→ Download→ (target execute file)], and then download the binary executable file to the evaluation board.

7. From the tabs at the top of the workspace, select [Debug→ Reset then execute], to run target board.

3.2 Host mode example

The following uses RX62N in an example to show how to setup the USB Basic Firmware Installation Guide and sample program to operate as a USB peripheral.

1. Select HwResourceForUSB

Delete the content of folder "*HwResourceForUSB*". Copy and paste the content of "*HwResourceForUSB_RX62N*" into "*HwResourceForUSB*". The folders named *HwResourceForUSB_XXX* are not used by any build configuration.

2. Select USB mode (Host/Peripheral)

Set macros *USB_FUNCSEL_USBIP0_PP* and *USB_FUNCSEL_USBIP1_PP* in the User Configuration file (r_usb_usrconfig.h) as follows.

```
//#define    USB_FUNCSEL_USBIP0_PP    USB_HOST_PP     // Host Mode
//#define    USB_FUNCSEL_USBIP0_PP    USB_PERI_PP     // Peripheral Mode
#define      USB_FUNCSEL_USBIP0_PP    USB_NOUSE_PP

#define      USB_FUNCSEL_USBIP1_PP    USB_HOST_PP     // Host Mode
//#define    USB_FUNCSEL_USBIP1_PP    USB_PERI_PP     // Peripheral Mode
//#define    USB_FUNCSEL_USBIP1_PP    USB_NOUSE_PP
```

3. Workspace

Double click “Fw.hws” to start up HEW. Then set the build configuration in the workspace to “Host”.

4. Generate an executable file

From the tabs at the top of the workspace, select [Build → Build all], then execute the build.

5. Connect to evaluation board

From the tabs at the top of the workspace, select [Debug→ Connect], and then connect the evaluation board to the emulator.

6. Download and execute the binary executable in the target

From the tabs at the top of the workspace, select [Debug→ Download→ (target execute file)], and then download the binary executable file to the evaluation board.

7. From the tabs at the top of the workspace, select [Debug→ Reset then execute], to run target board.

4. Peripheral sample program

4.1 Operation setting

4.1.1 Setting of Vendor ID/ Product ID

Changed file name: [r_usb_vendor_descriptor.c]

It is necessary to change Vendor ID/ Product ID that uses the system under development.

```
#define     USB_VENDORID        0x0000u     /* Vendor   ID */
#define     USB_PRODUCTID       0x0000u     /* Product  ID * /
```

4.1.2 Descriptor Table Generation

Changed file name[r_usb_vendor_descriptor.c]

In order for USB Basic Firmware Installation Guide to operate in peripheral mode, it is necessary to create a descriptor table that matches the class under development. (A sample table is included in r_usb_vendor_descriptor.c.)

The descriptor definitions comprise the following four types.

1) Standard Device Descriptor
   uint8_t usb_gpvendor_smpl_DeviceDescriptor[18]
2) Device Qualifier Descriptor
   uint8_t usb_gpvendor_smpl_QualifierDescriptor[10]
3) Configuration/Other_Speed_Configuration(*)/Interface/Endpoint
   uint8_t usb_gpvendor_smpl_ConfigurationH_1 [](*)
   uint8_t usb_gpvendor_smpl_ConfigurationF_1[]
4) String Descriptor
   uint8_t usb_gpvendor_string_descriptor1 []
   uint8_t usb_gpvendor_string_descriptor2 []
   uint8_t usb_gpvendor_string_descriptor3 []
   uint8_t usb_gpvendor_string_descriptor4 []
   uint8_t usb_gpvendor_string_descriptor5 []

* As for the RX600 series, this setting is not used for the Full-Speed operation..
PDCD must generate Configuration/Other_Speed_ConfigurationDescriptor when a USB reset is detected.

Note:

1. For details of each descriptor, see chapter 9 of the USB revision 2.0 specification
2. When making changes to descriptor definitions, it is also necessary to make changes to the pipe information table to match the endpoint descriptors.

4.1.3 PIPE Definition

Changed file name:[r_usb_vendor_descriptor.c]

PDCD must store appropriate pipe settings for the relevant class driver as a pipe information table. PDCD must register the pipe information table at the time of driver registration.

1) Pipe Information Table

A pipe information table comprises the following six items (uint16_t × 6).

1. Pipe window select register (address 0x64)
2. Pipe configuration register (address 0x68)
3. Pipe buffer designation register (address 0x6A)
4. Pipe maximum packet size register (address 0x6C)
5. Pipe period control register (address 0x6E)
6. FIFO port usage method

2) Definition of Pipes 1 to 9

The pipe definitions provided as a sample in USB Basic Firmware Installation Guide are configured as shown below.

The file r_usb_PSMPL_data.c contains macro definitions for the items that can be included in the information table are.

Example:

```
uint16_t usb_gpstd_SmplEpTbl1 [] = {                  ← Registered information table
    USB_PIPE1,                                        ← Pipe definition item 1
    USB_BULK|USB_BFREOFF|USB_DBLBOFF|USB_CNTMDOFF|USB_SHTNAKON|USB_DIR_P_OUT|USB_EP1,
                                                      ← Pipe definition item
    (uint16_t)USB_BUF_SIZE(512u) | USB_BUF_NUMB(8u),  ← Pipe definition item 3
    USB_SOFT_CHANGE,                                  ← Pipe definition item 4
    USB_IFISOFF | USB_IITV_TIME(0u),                  ← Pipe definition item 5
    USB_CUSE,                                         ← Pipe definition item 6
        :
        :
    USB_PDTBLEND                                      ← End of Pipe table
};
```

4.2 Function

The USB Basic Firmware Installation Guide peripheral sample program is configured with a sample vendor class driver and sample application and includes the following functions.

· Operation confirmation with “USBCommandVerifier.exe" is possible.

· Data transfer with USB Basic Firmware Installation Guide host sample program (see “5 Host Sample Program”)

· Works with the ANSI IO type API

(1) Operation confirmation with “USBCommandVerifier.exe"

The peripheral sample program operation can be confirmed operations based on the device framework test using the “USBCommandVerifier.exe” (USBCV） provided by the USB Implementers Forum (USB-IF). (Supported test items are listed in Chapter 9.)

(2) Data communications with a host sample program

The peripheral sample program can communicate with the host device running on theUSB-BASIC-F/W host sample program, transferring bulk and interrupt data.

The USB Basic Firmware Installation Guide peripheral sample program is configured with a sample vendor class driver and sample a

4.3 Operation of a peripheral sample program

The following describes an example of the peripheral sample program for non-OS operations.

(1). Initial Setting

When the device goes to the reset state, the PowerON_Reset_PC function in resetprg.c is called. The PowerON_Reset_PC function initializes the MCU, sections, the scheduler, etc., and then calls the usb_cstd_main_task function in main.c. After the USB module is set and tasks are registered in the usb_cstd_main_task function, the device returns to the static state.

Figure 4.1 shows the general flow of the initialization routine from reset state to static state.

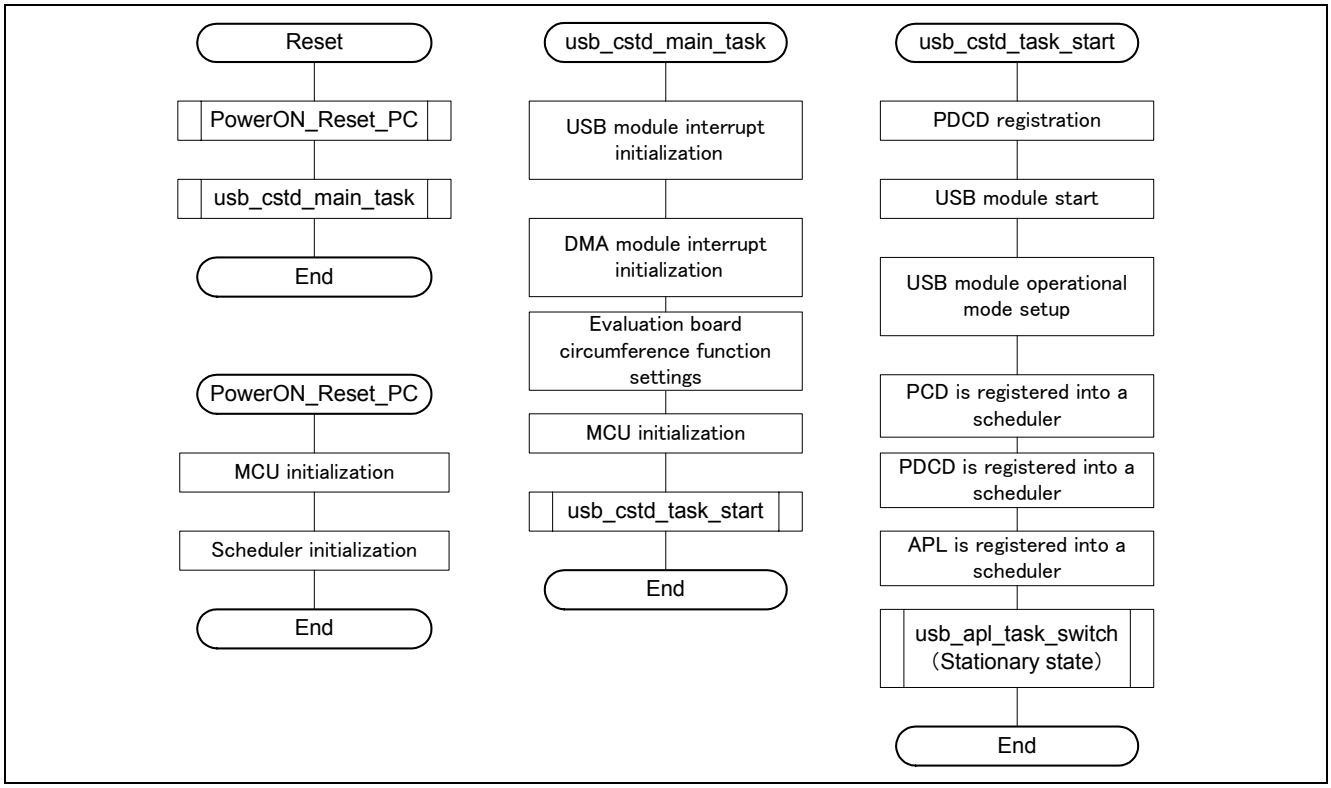

Figure 4.1 Peripheral Sample Program Intialization Overview

After initialization, the sample program calls the usb_apl_task_switch function and the device returns to the static state.

While the sample program is in the static state, enumeration and USB operation is triggered via interrupt.

Static state operations are described below.

(1) Checks for processing requests with scheduler.

(2) If processing requests are present, selects task with highest priority and sets task processing request flag.

(3) If task processing request flag is set, confirms the message of each task, and processes the tasks specified in each message

エラー! 参照元が見つかりません。 shows the program flow in the static state.

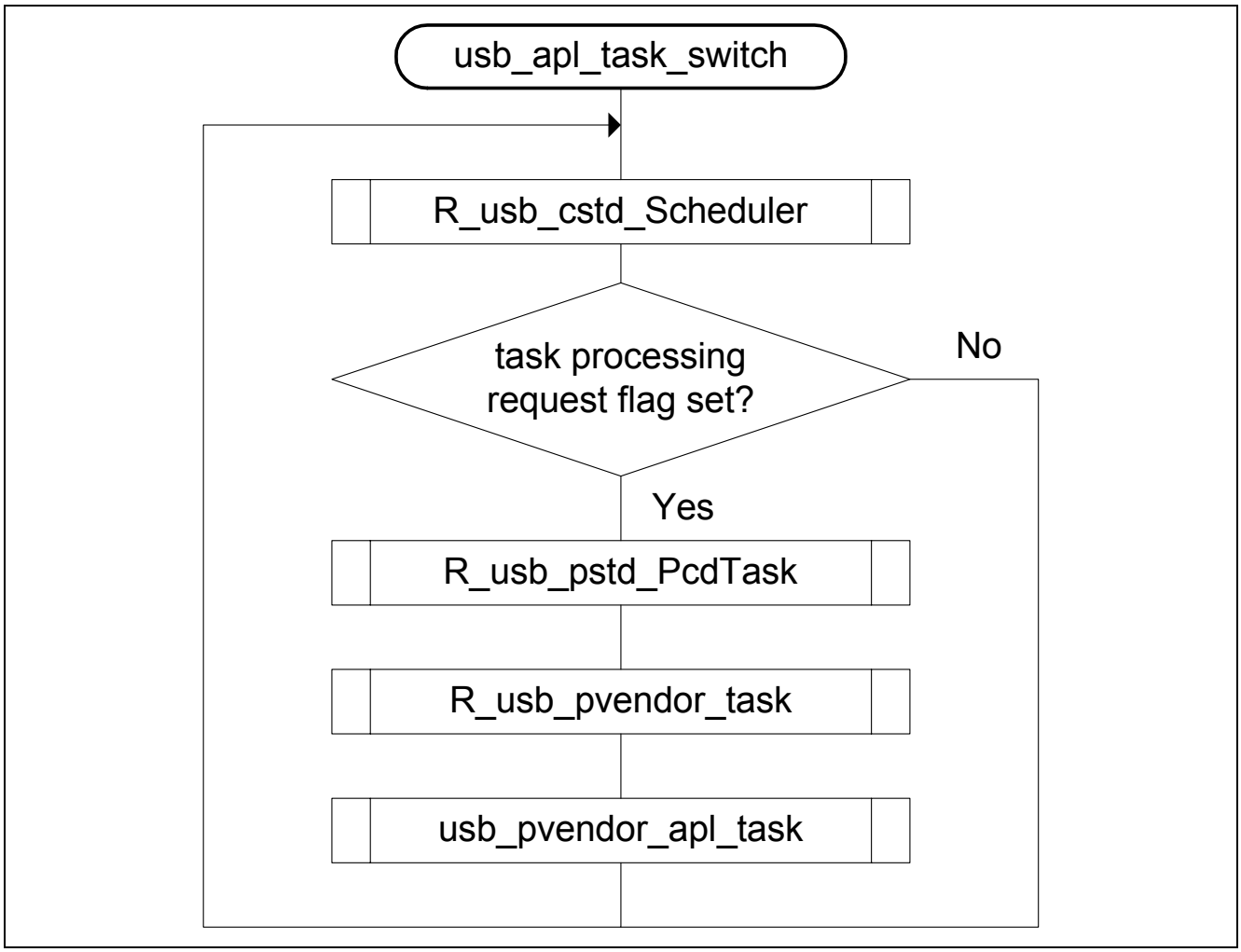

Figure 4.2 Static State Program Flow

(2). Sample Application Task (usb_pvendor_apl_task())

The sample application task performs open processing and data transmit/receive requests to the sample vendor driver, as described below.

· Open Processing

Checks if USB communications with USB host are enabled. If enabled, obtains file number.

· Data transmit/receive requests to sample vendor driver

After open processing is complete, the sample application task uses the file number obtained in the open processing to transmit/receive data to/from the sample driver.

(3). Sample Vendor Driver Task (usb_pvendor_task())

The sample vendor driver task notifies the PCD of the data transmit/receive request from the sample application task and performs the data transmit/receive operation.

4.3.1 Sample Application Sequence Outline

Figure 4.3 shows Sequence Outline for Periferal Sample application.

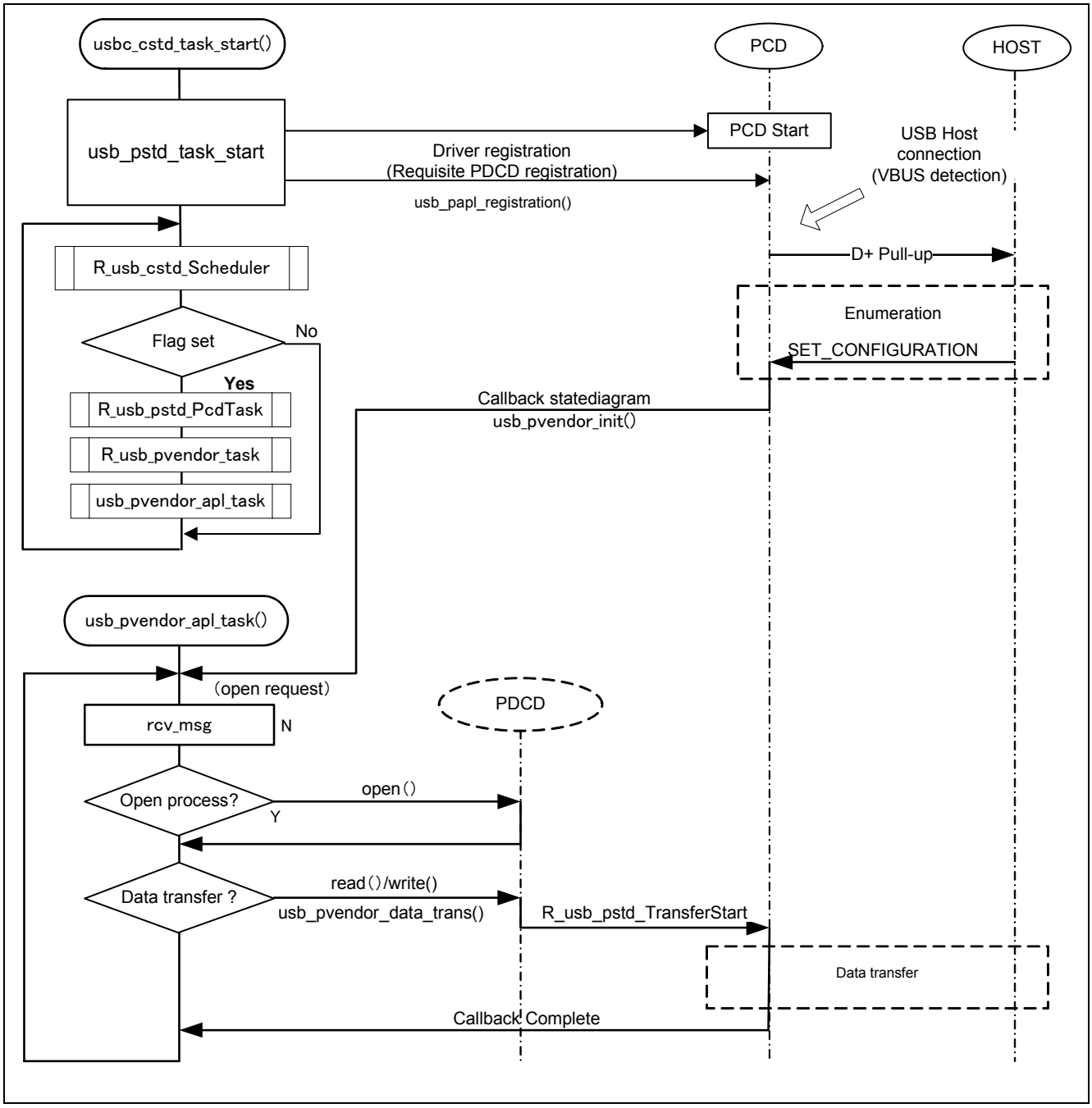

Figure 4.3 Sequence Outline for Periferal Sample application

(1) Register the PDCD

The structure used to register information in PDCD is described below. The call-back function is executed when the device state changes, etc.

The sample application is described in the usb_papl_registration in r_usb_vendor_papl.c.

```
typedef struct USB_PCDREG
{
        uint16_t            **pipetbl;      /* Pipe Define Table address */
        uint8_t             *devicetbl;     /* Device descriptor Table address */
        uint8_t             *qualitbl;      /* Qualifier descriptor Table address */
        uint8_t             **configtbl;    /* Configuration descriptor Table address */
        uint8_t             **othertbl;     /* Other configuration descriptor Table address */
        uint8_t             **stringtbl;    /* String descriptor Table address */
        USB_CB_INFO_t       classinit;      /* Driver init */
        USB_CB_INFO_t       devdefault;     /* Device default */
        USB_CB_INFO_t       devconfig;      /* Device configured */
        USB_CB_INFO_t       devdetach;      /* Device detach */
        USB_CB_INFO_t       devsuspend;     /* Device suspend */
        USB_CB_INFO_t       devresume;      /* Device resume */
        USB_CB_INFO_t       interface;      /* Interface changed */
        USB_CB_TRN_t ctrltrans;             /* Control Transfer */
} USB_PCDREG_t;
```

Table 4-1 Member of USB_PCDREG_t structure

type	member	Description
uint16_t	**pipetbl	Register the address of the pipe information table.
uint8_t	*devicetbl	Register the device descriptor address.
uint8_t	*qualitbl	Register the device qualifier descriptor address.
uint8_t	**configtbl	Register the address of the configuration descriptor address table.
uint8_t	**othertbl	Register the address of the other speed descriptor address table.
uint8_t	**stringtbl	Register the address of the string descriptor address table.
USB_CB_INFO_t	classinit	Register the function to be started when initializing PDCD. It is called at registration.
USB_CB_INFO_t	devdefault	Register the function to be started when transitioning to the default state. It is called when a USB reset is detected.
USB_CB_INFO_t	devconfig	Register the function to be started when transitioning to the configured state. It is called in the SET_CONFIGURATION request status stage.
USB_CB_INFO_t	devdetach	Register the function to be started when transitioning to the detach state. It is called when a detached condition is detected.
USB_CB_INFO_t	devsuspend	Register the function to be started when transitioning to the suspend state. It is called when a suspend condition is detected.
USB_CB_INFO_t	devresume	Register the function to be started when transitioning to the resume state. It is called when a resume condition is detected.
USB_CB_INFO_t	interface	Register the function to be started when an interface change occurs. It is called in the SET_INTERFACE request status stage.
USB_CB_TRN_t	ctrltrans	Register the function to be started when a user-initiated control transfer occurs.

4.3.2 Example of Creating a Sample Application

The following is an example of embedding a USB Basic Firmware Installation Guide data transfer sample application. For more details, refer to r_usb_vendor_capl.c and r_usb_vendor_papl.c.

(1) Example of creating a main task

```
"r_usb_vendor_capl.c"

void usb_cstd_task_start( void )
{
        usb_cstd_IdleTaskStart();       /* Idle Task start */

        usb_hstd_task_start();          /* Task start for USB Host (Start the HCD, Register the HDCD) */
        usb_pstd_task_start();          ← Task start for USB Peripheral (Start the PCD, Register the PDCD)
        usb_apl_task_switch();          ← Mainloop
}
```

Figure.4.4 Example of coding a Peripheral Application (1)

(2) Example of creating register the PCDC

"_usb_vendor_papl.c"

```
void usb_pstd_task_start( void )
{
        USB_UTR_t       utr;
        USB_UTR_t       *ptr;

        ptr = &utr;
        ptr->ip = USB_PERI_USBIP_NUM;
        if( USB_NOUSE_PP != ptr->ip )
        {
                ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );

                usb_papl_registration( ptr );       /* Peripheral Application Registration */
                R_usb_pvendor_driver_start();       /* Peripheral Class Driver Task Start Setting */
                usb_pstd_usbdriver_start( ptr );    /* Peripheral USB Driver Start Setting */
                usb_papl_task_start( ptr );         /* Peripheral Application Task Start Setting */
        }
}
```

Register the sample PDCD driver (*)

*:You can also create/register thePDCD to fit your user system.

Figure.4.5 Example of coding a Peripheral Application (2)

(3) Example of coding a Peripheral sample application

"r_usb_vendor_capl.c"

```
void usb_apl_task_switch(void)
{
/* Condition compilation by the difference of the operating system */
#if USB_FW_PP == USB_FW_NONOS_PP
        while( 1 )
        {
                /* Scheduler */
                R_usb_cstd_Scheduler();

                if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
                {
(skip)
#ifdef  USB_PERI_MODE_PP
                        R_usb_pstd_PcdTask((USB_VP_INT)0);          /* PCD Task */
                        R_usb_pvendor_task((USB_VP_INT)0);          /* Peripheral Vendor class Task */
                        usb_pvendor_apl_task((USB_VP_INT)0);        /* Peripheral Class Sample Task */
#endif /* USB_PERI_MODE_PP */
                }
                else
                {
                        /* Idle Task (sleep sample) */
                        R_usb_cstd_IdleTask(0);
                }
        }
#endif /* USB_FW_PP == USB_FW_NONOS_PP */
}
```

Sample application Task
(OPEN process,
Data transmission sample application)

Figure.4.6 Example of coding a Peripheral Application (3)

5. Host Sample Program

5.1 Oparetion seting

5.1.1 Target Peripheral List

Changed file name::[r_usb_vendor_hapi.c]

A TPL is generated for each device class. In project, vendor IDs and product IDs are registered as sets. Register all supported sets in the relevant device class.

```
const uint16_t tpl[] = {
        3,                              /* Number of list */
        0,                              /* Reserved */
        0xXXXX,                         /* Vendor ID */
        0xXXXX,                         /* Product ID */
        0xYYYY,                         /* Vendor ID */
        0xYYYY,                         /* Product ID */
        0xZZZZ,                         /* Vendor ID */
        0xZZZZ,                         /* Product ID */
};
```

When all vendor IDs and product IDs in the device class are supported, do not specify specific vendor IDs and product IDs. Instead, register USB_NOVENDOR and USB_NOPURODUCT as a set.

```
const uint16_t tpl[] = {
        1,                              /* Number of list */
        0,                              /* Reserved */
        USBC_NOVENDOR,                  /* Vendor ID */
        USBC_NOPRODUCT,                 /* Product ID */
};
```

5.1.2 Pipe Definition

Changed file name:[¥VENDOR¥r_usb_nendor_hdefep.c]

HDCD must store appropriate pipe settings for the relevant class driver as a pipe information table. Update the pipe information table to match the user system. For more details, refer to the Renesas USB MCU and USB ASSP USB Basic Firmware Application Note (Document No. R01AN0512EJ).

Note that the pipe information table must be updated to match the connected device. The USB Basic Firmware Installation Guide is equipped with R_usb_hstd_ChkPipeInfo() to analyze the descriptor it receives from the device (Endpoint Descriptor) and update the data in the pipe information table. It also comes with R_usb_hstd_SetPipeInfo () for setting the updated information in the pipe information table. The user can use these functions to update the pipe information table to match the connected device.

5.2 Function

The USB Basic Firmware Installation Guide host sample program is configured with the vendor class driver and sample application and includes the following functions.

· Data transfer with USB Basic Firmware Installation Guide peripheral sample program (see 5 Host Sample Program)
· ANSI IO compliant APIs

5.3 Host Sample Program Operations

The following describes an example of the host sample program for non-OS operations.

(1). Initial Setting

When the device goes to the reset state, the PowerON_Reset_PC function in resetprg.c is called.

The PowerON_Reset_PC function initializes the MCU, sections, the scheduler, etc., and then calls the usb_cstd_main_task function in main.c. After the USB module is set and tasks are registered in the usb_cstd_main_task function, the device returns to the static state.

Figure 5.1 shows the general flow of the initialization routine from reset state to static state.

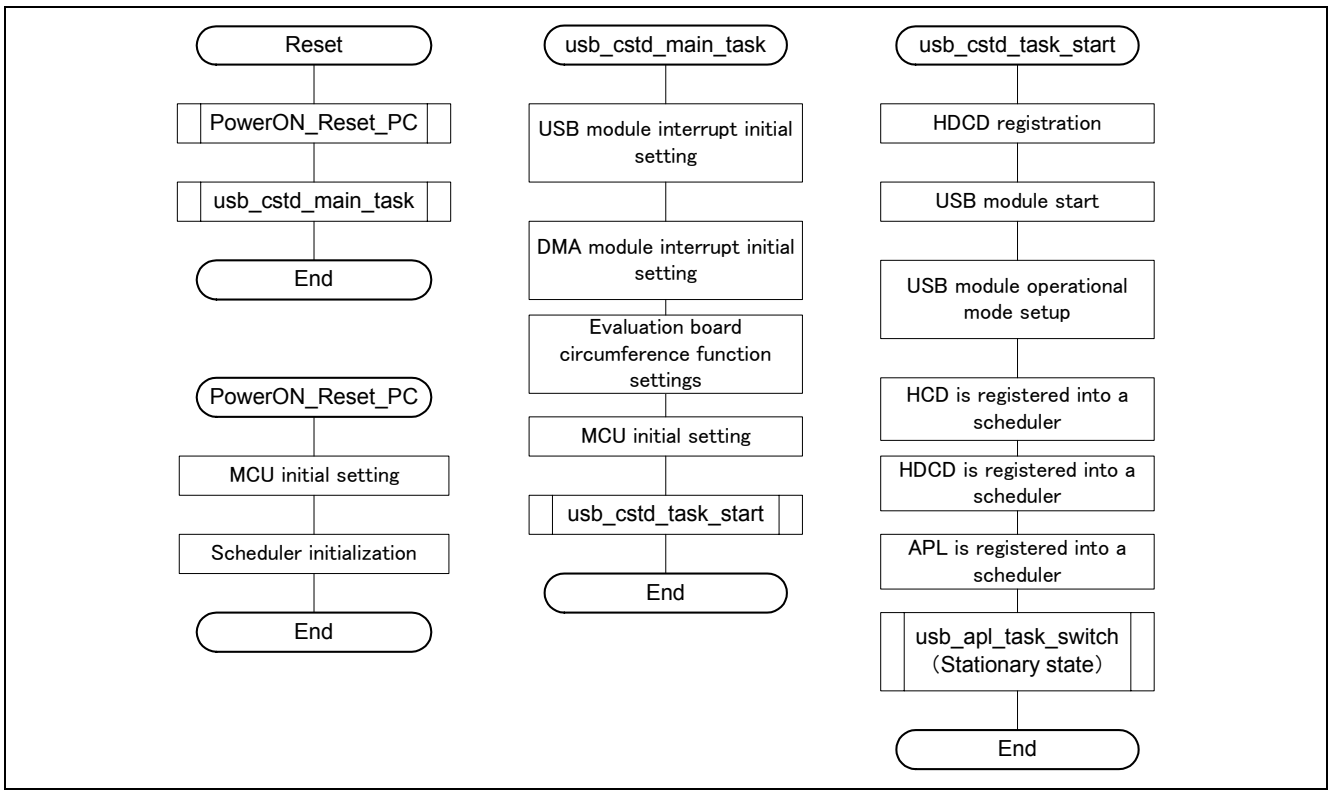

Figure 5.1 Host Sample Program Intialization Overview

After initialization, the sample program calls the usb_apl_task_switch function and the device returns to the static state.

(1) Checks for processing requests with scheduler.

(2) If processing requests are present, selects task with highest priority and sets task processing request flag

(3) If task processing request flag is set, confirms the message of each task, and processes the tasks specified in each message

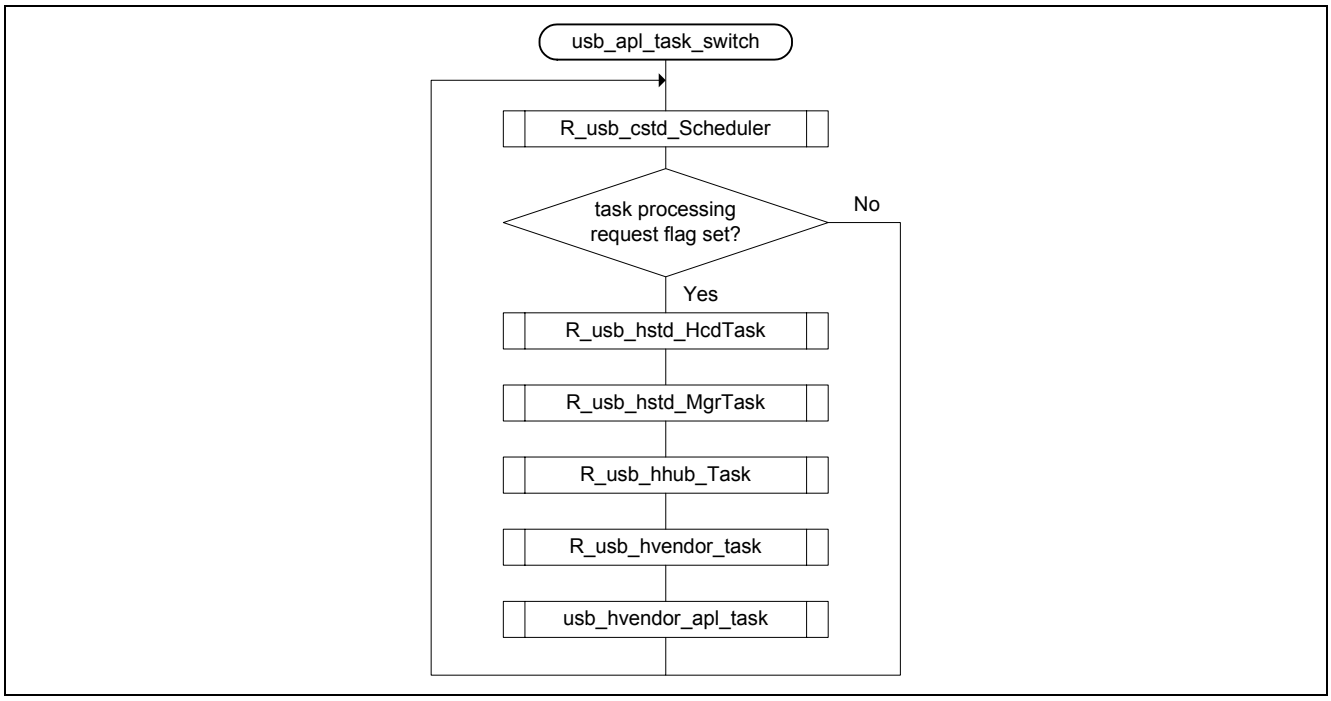

Figure 5.2 Static State Program Flow

(2). Sample Application Task (usb_hvendor_apl_task())

The sample application task performs open processing and data transmit/receive requests for the sample driver, as described below.

· Open Processing

Checks if USB communications with USB device are enabled. If enabled, obtains file number.

· Data transmit/receive requests for sample driver

After open processing is complete, the sample application task uses the file number obtained in the open processing to transmit/receive data to/from the sample driver.

(3). Sample Driver Task (usb_hvendor_task())

The sample driver task notifies the HCD of the data transmit/receive request from the sample application task and performs the data transmit/receive operation.

5.3.1 Sample Application Sequence Outline

Figure .5.3 shows Sequence Outline for Host Sample application.

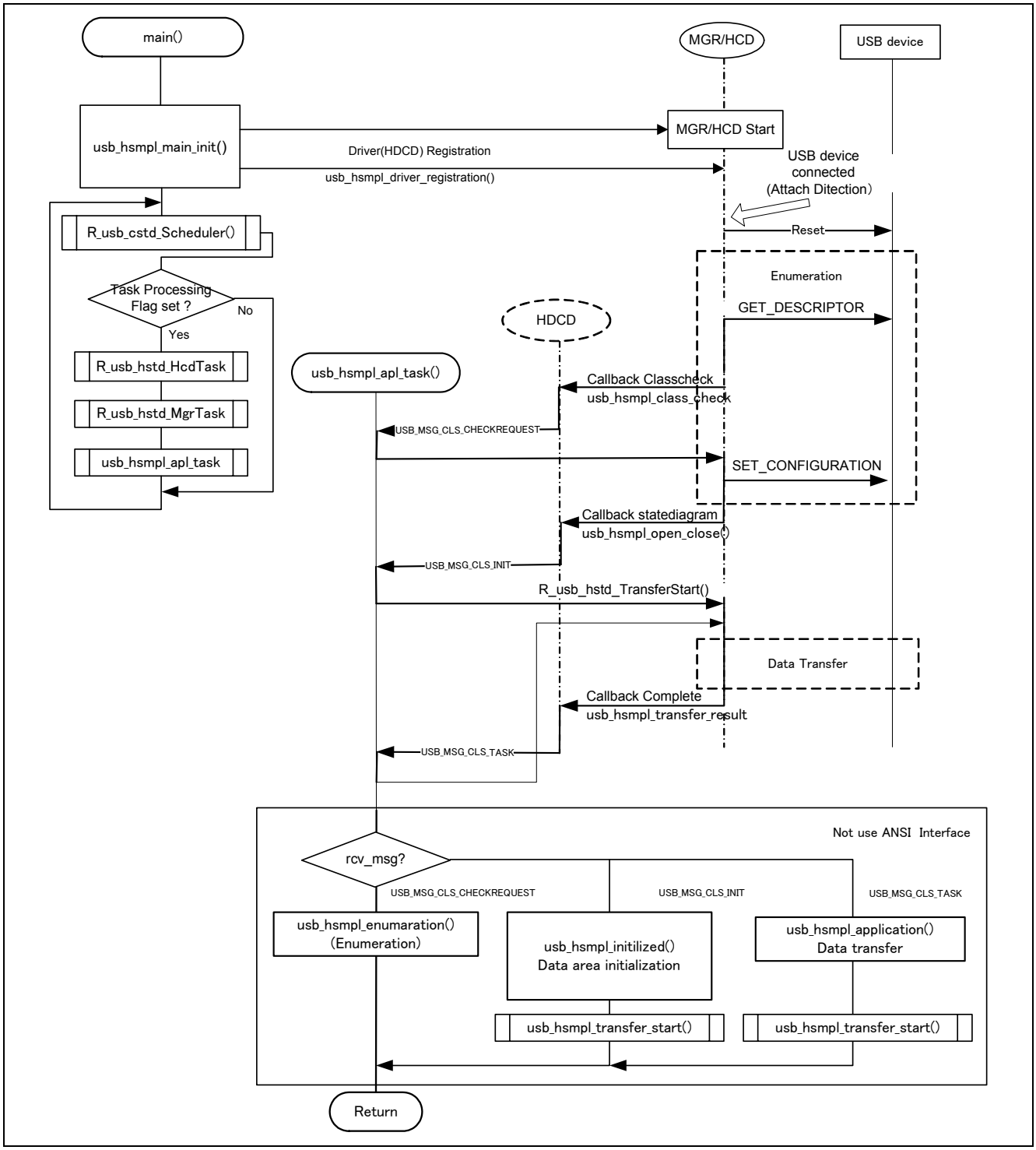

Figure .5.3 Sequence Outline for Host Sample application

(1) Register the HDCD

The structure used to register information in HDCD is described below. The sample application is described in the usb_hapl_registration function in r_usb_vendor_hapl.c.

```
typedef struct USB_HCDREG
{
        uint16_t            rootport;       /* Root port */
        uint16_t            devaddr;        /* Device address */
        uint16_t            devstate;       /* Device state */
        uint16_t            ifclass;        /* Interface Class */
        uint16_t            *tpl;           /* Target peripheral list (Vendor ID, Product ID) */
        uint16_t            *pipetbl;       /* Pipe Define Table address */
        USB_CB_INFO_t       classinit;      /* Driver init */
        USB_CB_CHECK_t      classcheck;     /* Driver check */
        USB_CB_INFO_t       devconfig;      /* Device configured */
        USB_CB_INFO_t       devdetach;      /* Device detach */
        USB_CB_INFO_t       devsuspend;     /* Device suspend */
        USB_CB_INFO_t       devresume;      /* Device resume */
        USB_CB_INFO_t       overcurrent;    /* Device over current */
} USB_HCDREG_t;
```

Table 5.1 Member of USB_HCDREG_t Structure

Type	Variable Name	Description
uint16_t	rootport	Used by HCD. The connected port number is registered.
uint16_t	devaddr	Used by HCD. The device address is registered.
uint16_t	devstate	Used by HCD. The device connection state is registered.
uint16_t	ifclass	Register the interface class code for HDCD operation.
uint16_t	*tpl	Register the target peripheral list for HDCD operation.
uint16_t	*pipetbl	Register the address of the pipe information table.
USB_CB_INFO_t	classinit	Register the function started at driver registration.
USB_CB_CHECK_t	classcheck	Register the function started at HDCD checking operation. It is called when a TPL match occurs.
USB_CB_INFO_t	devconfig	Register the function to be started when transitioning to the configured state. It is called in the SET_CONFIGURATION request status stage.
USB_CB_INFO_t	devdetach	Register the function to be started when transitioning to the detach state.
USB_CB_INFO_t	devsuspend	Register the function to be started when transitioning to the suspend state.
USB_CB_INFO_t	devresume	Register the function to be started when transitioning to the resume state.
USB_CB_INFO_t	overcurrent	Register the function started when overcurrent detection occurs.

5.3.2 Example of Creating a Host Application

The following is an example of embedding a USB Basic Firmware Installation Guide data transfer sample application. . For more details, refer to r_usb_vendor_capl.c and r_usb_vendor_hapl.c .

(1) Example of creating a main task

```
"r_usb_vendor_capl.c"

void usb_cstd_task_start( void )
{
        usb_cstd_IdleTaskStart();      /* Idle Task スタート */

        usb_hstd_task_start();          ← Task start for USB Host (Start the HCD, Register the HDCD)
        usb_pstd_task_start();          ← Task start for USB Peripheral (Start the PCD, Register the PDCD)
        usb_apl_task_switch();          ← Mainloop
}
```

Figure.5.4 Example of coding a Host Application(1)

(2) Register the HDCD

```
"r_usb_vendor_hapl.c"

void usb_hstd_task_start( void )
{
        USB_UTR_t       utr;
        USB_UTR_t       *ptr;

        ptr = &utr;
        ptr->ip = USB_PERI_USBIP_NUM;
        if( USB_NOUSE_PP != ptr->ip )
        {
                ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );

                R_usb_hvendor_driver_start();          /* Host Class Driver Task Start Setting */
                usb_hstd_usbdriver_start( ptr );       /* Host USB Driver Start Setting */
                usb_hapl_registration( ptr );          /* Host Application Registration */
                usb_hapl_task_start( ptr );            /* Host Application Task Start Setting */
        }
}
```

Register the sample HDCD driver (*)

*:You can also create/register the HDCD to fit your user system.

Figure..5.5 Example of coding a Host Application (2)

(3) Example of coding a Host sample application

"r_usb_vendor_capl.c"

```
void usb_apl_task_switch(void)
{
/* Condition compilation by the difference of the operating system */
#if USB_FW_PP == USB_FW_NONOS_PP
        while( 1 )
        {
                /* Scheduler */
                R_usb_cstd_Scheduler();

                if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
                {
#if USB_FW_PP == USB_FW_NONOS_PP
        while( 1 )
        {
                /* Scheduler */
                R_usb_cstd_Scheduler();

                if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
                {
#ifdef  USB_HOST_MODE_PP
                        R_usb_hstd_HcdTask((USB_VP_INT)0);              /* HCD Task */
                        R_usb_hstd_MgrTask((USB_VP_INT)0);              /* MGR Task */
                        R_usb_hhub_Task((USB_VP_INT)0);                 /* HUB Task */
                        R_usb_hvendor_task((USB_VP_INT)0);              /* Host Vendor class Task */
                        usb_hvendor_apl_task((USB_VP_INT)0);        /* Host Class Sample Task */
#endif /* USB_HOST_MODE_PP */
(Skip)
```

Figure.5.6 Example of coding a Host Application (3)

6. Software Configuration

In peripheral mode, USB Basic Firmware Installation Guide comprises the peripheral driver (PCD), and the application (APL). PDCD is the class driver and not part of the USB-BASIC-F/W. See Table 6.1. In host mode, USB Basic Firmware Installation Guide comprises the host driver (HCD), the manager (MGR), the hub class driver (HUBCD) and the application (APL). HDD and HDCD are not part of the USB-BASIC-F/W, see Table 6.1.

The peripheral driver (PCD) and host driver (HCD) initiate hardware control through the hardware access layer according to messages from the various tasks or interrupt handler. They also notify the appropriate task when hardware control ends, of processing results, and of hardware requests.

Manager manages the connection state of USB peripherals and performs enumeration. In addition, manager issues a message to host driver or hub class driver when the application changes the device state. Hub class driver is sample program code for managing the states of devices connected to the down ports of the USB hub and performing enumeration.

The customer will need to make a variety of customizations, for example designating classes, issuing vendor-specific requests, making settings with regard to the communication speed or program capacity, or making individual settings that affect the user interface.

In addition, PDCD and HDCD need to be created by user. Please refer to the following sample program about how to create PDCD and HDCD.

PDCD WorkSpace¥VENDOR¥r_usb_vendor_pdriver.c
HDCD WorkSpace¥VENDOR¥r_usb_vendor_hdriver.c

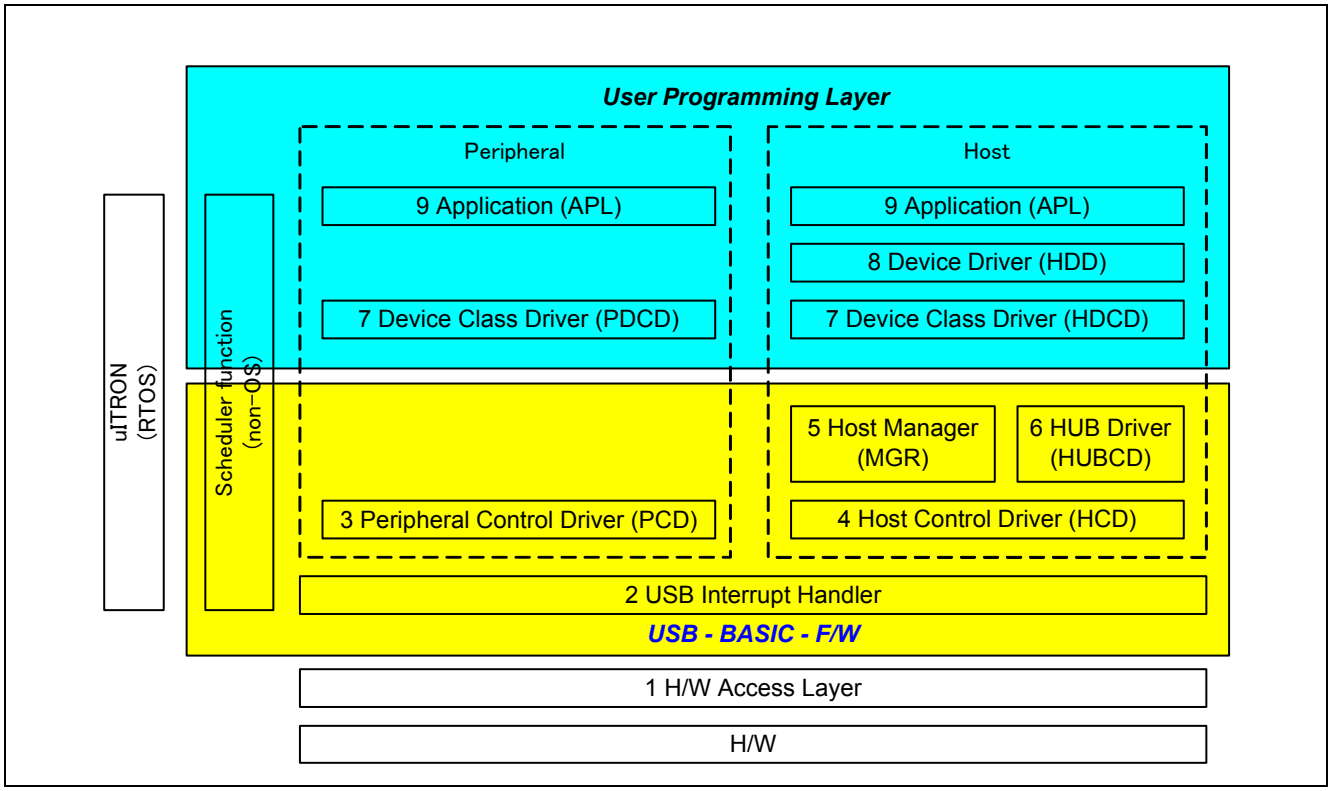

Fig 6.1 Software Configuration of USB Basic Firmware Installation Guide

Table 6.1 Task Functions

No	Module Name	Function
1	USB interrupt handler void usb_cstd_UsbHandler(void)	USB interrupt handler (USB packet transmit/receive end and special signal detection)
2	Peripheral driver(PCD) void usb_pstd_PcdTask(USBC_VP_INT_t)	Hardware control in peripheral function mode Peripheral transaction management
3	Host driver (HCD) void usb2_hstd_HcdTask(USBC_VP_INT_t)	Hardware control in host function mode Host transaction management
4	Host manager (MGR) void usb2_hstd_MgrTask(USBC_VP_INT_t)	Device state management Enumeration HCD/HUBCD control message determination
5	Hub class driver (HUBCD) void usb2_hhub_Task(USBC_VP_INT_t)	HUB down port device state management HUB down port enumeration
6	USB Peripheral Device class driver(PDCD)	Provided by the customer as appropriate for the system. Refer to Chapter.3.2 to how to provide.
7	USB Host Device class driver (HDCD)	Provided by the customer as appropriate for the system. Refer to Chapter.3.3 to how to provide.
8	Application (APL)	Provided by the customer as appropriate for the system.

6.1 List of Files

6.1.1 Folder Structure

The folder structure in which the files are provided in USB Basic Firmware Installation Guide(non-OS & RTOS) is shown below. The USB-BASIC-F/W includes a sample(vendor) class driver, application and hardware resource sample code.

The source codes dependent on each device and evaluation board are stored in each hardware resource folder (HwResourceForUSB_***devicename***).

For the file, refer to each configuration setting. For more details, refer to the Renesas USB MCU and USB ASSP USB Basic Firmware Application Note (Document No. R01AN0512EJ).

```
Workspace
(USB Basic Firmware Installation Guide)    [Common USB code that is used by all USB firmware]
    ＋―――USBSTDFW
    │          ＋―――non-OS/RTOS
    │          │          ＋―――class
    │          │          │          ＋―――hubd      Sample hub driver
    │          │          │          ＋―――SMPL      USB standard request sample
    │          │          ＋―――USB20
    │          │          ＋―――HCD       Host control driver
    │          │          ＋―――PCD       Peripheral control driver
    │          │          ＋―――LIB       Common library
    │          ＋include                  Common header file
(ANSI-C File I/O System Calls)
    ＋―――ANSI [open(), close(), read(), write(), etc of the USB class driver]
(HW Setting) [Hardware access layer; to manipulate the MCU's USB register]
    ＋―――HwResourceForUSB                  Selected H/W resource
    ＋―――HwResourceForUSB_RX62N            Hardware resource for RX62N/RX621 Group
    ＋―――HwResourceForUSB_RX62N_597assp    Hardware resource for RX62N + R8A66597
    ＋―――HwResourceForUSB_RX63N            Hardware resource for RX63N/RX631 Group
    ＋―――HwResourceForUSB_RX630            Hardware resource for RX630 Group
(Sample Code) [Class driver and user application]
    ＋―――SmplMain                          Sample application
    ＋―――VENDOR                            Vendor Class Driver
(uITRON) [uItron OS code]
    ＋―――RI600_4                           RTOS only
```

Website and Support

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

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Apr.15.11	—	First edition issued
1.10	Aug.24.11	Cover, Other (2,3,4,5,6, 7,9,11,12, 21,32)	- Modified Document title RX62N Group -> RX600 Series, Short Sheet -> Installation Guide - Add RX630 Group to Target Device (The content according to the Target device addition is changed.)
2.00	Jun. 01.2012	—	Revision of the document by firmware upgrade

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

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

SALES OFFICES Renesas Electronics Corporation http://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: +49-211-65030, 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.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
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

Colophon 1.1