RZ/T1 Group

R01AN2631EJ0130
Rev.1.30
Dec 07, 2017

USB Peripheral Communications Device Class Driver (PCDC)

Introduction

This application note describes USB Peripheral Communication Device Class Driver. This module performs hardware control of USB communication. It is referred to below as the USB-BASIC-F/W. After a while calls this sample software PCDC.

The sample program of this application note is created based on "RZ/T1 group Initial Settings Rev.1.30". Please refer to "RZ/T1 group Initial Settings application note (R01AN2554EJ0130)" about operating environment.

Target Device

RZ/T1 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

1. Universal Serial Bus Revision 2.0 specification
2. USB Class Definitions for Communications Devices Revision 1.2
3. USB Communications Class Subclass Specification for PSTN Devices Revision 1.2
   http://www.usb.org/developers/docs/
4. RZ/T1 Group User's Manual: Hardware (Document No.R01UH0483EJ0130)
5. RZ/T1 Group Initial Settings (Document No.R01AN2554EJ0130)
6. USB Peripheral Basic Firmware (Document No.R01AN2630EJ0130)

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

USB Devices Page
http://www.renesas.com/prod/usb/

Contents

1. Overview

The PCDC, when used in combination with the USB-BASIC-F/W, operates as a USB peripheral communications device class driver. The PCDC conforms to the abstract control model of the USB communication device class specification and enables communication with a USB host.

This module supports the following functions.

- Data transfer to and from a USB host
- Response to CDC class requests
- Provision of communication device class notification transmit service

Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

ACM	:	Abstract Control Model. This is the USB interface subclass used for virtual COM ports, based in the old V.250 (AT) command standard. See PSTN below.
APL	:	Application program
CDC	:	Communications Devices Class
CDCC	:	Communications Devices Class Communications Interface Class
CDCD	:	Communications Devices Class Data Class Interface
H/W	:	Renesas USB device
PCD	:	Peripheral control driver of USB-BASIC-F/W
PCDC	:	Communications Devices Class for peripheral
PSTN	:	Public Switched Telephone Network, contains the ACM (above) standard.
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB Peripheral Basic firmware for Renesas USB device (non-OS)

2. Software Configuration

Figure 2-1 shows the block diagram of PCDC, and Table 2.1 lists the modules.

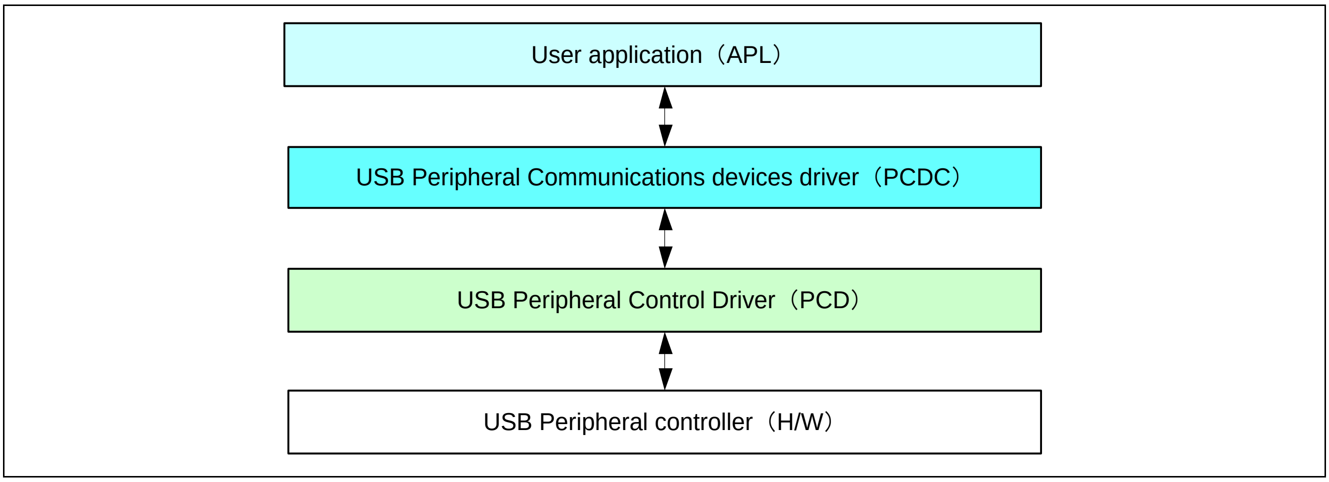


Figure 2-1 Source Code Block Diagram

Table 2.1 Modules

Module	Description
APL	User application program. (Please prepare for your system)
PCDC	Peripheral Communications Device Class Driver Sends requests from the APL for requests and data communication involving the CDC to the PCD.
PCD	USB peripheral hardware control driver. (USB-BASIC-F/W)

3. Peripheral Communications Device Class Driver (PCDC)

3.1 Basic Functions

The PCDC conforms to the Abstract Control of the CDC PSTN Subclass.

The main functions of the PCDC as follows:

1. Respond to functional inquiries from the USB Host
2. Respond to class requests from the USB Host
3. Data communication with the USB Host
4. Notify the USB Host of serial communication errors

3.2 Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for earlier modems. The class requests and class notifications supported are listed below.

3.2.1 Class Requests (Host to Peripheral)

Table 3.1 shows CDC class requests, and whether they are supported.

Table 3.1 CDC class requests

Request	Code	Description	Supported
SendEncapsulatedCommand	0x00	Transmits AT commands, etc., defined by the protocol.	No
GetEncapsulatedResponse	0x01	Requests a response to a command transmitted by SendEncapsulatedCommand.	No
SetCommFeature	0x02	Enables or disables features such as device-specific 2-byte code and country setting.	No
GetCommFeature	0x03	Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting.	No
ClearCommFeature	0x04	Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting.	No
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).	**Yes**
GetLineCoding	0x21	Acquires the communication line setting state.	**Yes**
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.	**Yes**
SendBreak	0x23	Transmits a break signal.	No

Yes : Implemented No : Not implemented(Stall response)

For details concerning the Abstract Control Model requests, refer to Table 11, “Requests - Abstract Control Model” in “USB Communications Class Subclass Specification for PSTN Devices”, Revision 1.2.

3.2.2 Data Format of Class Requests

The data format of the class requests supported by the class driver software is described below.

(1). SetLineCoding

This is the class request the host transmits to perform the UART line setting.

The SetLineCoding data format is shown below.

Table 3.2 SetLineCoding Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x00	0x00	0x07	Line Coding Structure See Table 3.3 Line Coding Format

Table 3.3 Line Coding Format

Offset	Field	Size	Value	Description
0	DwDTERate	4	Number	Data terminal speed (bps)
4	BcharFormat	1	Number	Stop bits 0 - 1 stop bit 1 - 1.5 stop bits 2 - 2 stop bits
5	BparityType	1	Number	Parity 0 - None 1 - Odd 2 - Even
6	BdataBits	1	Number	Data bits (5, 6, 7, 8)

(2). GetLineCoding

This is the class request the host transmits to request the UART line state.

The GetLineCoding data format is shown below.

Table 3.4 SetLineCoding Format

bmRequestType	bRequest	wValue	WIndex	wLength	Data
0xA1	GET_LINE_CODING (0x21)	0x00	0x00	0x07	Line Coding Structure See table 3.3, Line Coding Structure Format

(3). SetControlLineState

This is a class request that the host sends to set up the signal for flow controls of UART.

This software does not support RTS/DTR control.

The SET_CONTROL_LINE_STATE data format is shown below.

Table 3.5 SET_CONTROL_LINE_STATE Format

bmRequestType	bRequest	WValue	wIndex	wLength	Data
0x21	SET_CONTROL_ LINE_STATE (0x22)	Control Signal Bitmap See Table 3.6, Control Signal Bitmap Format	0x00	0x00	None

Table 3.6 Control Signal Bitmap

Bit Position	Description	
D15 to D2	Reserved (reset to 0)	
D1	DCE transmit function control	0 - RTS Off 1 - RTS On
D0	Notification of DTE ready state	0 - DTR not present 1 - DTR present

3.2.3 Class Notifications (Peripheral to Host)

Whether or not a class notification is supported is shown in Table 3.7.

Table 3.7 CDC Class Notifications

Notification	Code	Description	Supported
NETWORK_CONNECTION	0x00	Notification of network connection state	No
RESPONSE_AVAILABLE	0x01	Response to GET_ENCAPSLATED_RESPONSE	No
SERIAL_STATE	0x20	Notification of serial line state	Yes

(1). SerialState

The host is notified of the state when a change in the UART is detected.

notification is performed when a change from normal state to error is detected. However, notification is not conti This software supports the detection of overrun, parity and framing errors. A state nually transmitted when an error is continually detected.

The SerialState data format is shown below.

Table 3.8 SerialState Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x00	0x00	0x02	UART State bitmap See Table 3.9 UART state bitmap format

Table 3.9 UART state bitmap format

Bits	Field	Description
D15～D7		Reserved
D6	bOverRun	Overrun error detected
D5	bParity	Parity error detected
D4	bFraming	Framing error detected
D3	bRingSignal	INCOMING signal (ring signal) detected
D2	bBreak	Break signal detected
D1	bTxCarrier	Data Set Ready: Line connected and ready for communication
D0	bRxCarrier	Data Carrier Detect: Carrier detected on line

3.3 PC Virtual COM-port Usage

The CDC device can be used as a virtual COM port when operating in Windows OS.

Use a PC running Windows OS, and connect an evaluation board. After USB enumeration, the CDC class requests *GetLineCoding* and *SetControlLineState* are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as "HyperTerminal" which comes standard with Windows OS. When changing settings of the serial port in the Windows terminal application, the communication line setting is propagated to the firmware via the class *request SetLineCoding*.

Data input (or file transmission) from the terminal app window is transmitted to the evaluation board. Data transfer from the evaluation board is output in the terminal app window.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

The received data is outputed on the terminal when the data less than Maximum packet size is received.

3.4 API

All API calls and their supporting interface definitions are located in r_usb_pcdc_if.h.

Please modify r_usb_pcdc_config.h when User sets the module configuration option.

Table 3-10 shows the option name and the setting value.

Table 3.10 Configuration options of PCDC

Define name	Default value	Description
USB_PMSC_USE_PIPE_IN	USB_PIPE1	Pipe number of Bulk-IN transfer
USB_PMSC_USE_PIPE_OUT	USB_PIPE2	Pipe number of Bulk-OUT transfer
USB_PCDC_USE_PIPE_STATUS	USB_PIPE6	Pipe number of Interrupt-IN transfer

Table 3.11 shows all the PCDC API functions.

Table 3.11 PCDC API Functions

Function	Description
R_usb_pcdc_SendData	Sends a data transmit request message to PCDC task.
R_usb_pcdc_ReceiveData	Sends a data receive request message to PCDC task.
R_usb_pcdc_SerialStateNotification	Sends a class notification "SerialState" to the PCDC task.
R_usb_pcdc_ctrltrans	Control transfer processing for CDC

3.4.1 R_usb_pcdc_SendData

Transfer USB data

Format

void	R_usb_pcdc_SendData (uint8_t* table, uint32_t size, USB_UTR_CB_t complete)

Argument

*table	Pointer to buffer containing data to transmit
size	Transfer size
complete	Process completion callback function

Return Value

—

Description

This function transfers specified size from the specified address.
When the transmission is done, the callback function 'complete' is called.

Note

The USB transmit process results are found in the callback function's arguments.
See "USB Communication Structure" (USB_UTR_t) in the USB-BASIC-FW application note.

Example

```
void usb_apl_task( void )
{
  uint8_t    send_data[] = {0x01,0x02,0x03,0x04,0x05};     /* USB send data */
  uint32_t   size = 5;                                      /* Data size */

  R_usb_pcdc_SendData(send_data, size, &usb_complete);
}

/* Callback function */
void usb_complete(USB_UTR_t *mess)
{
  /* Processing at the time of the completion of USB transmitting */
}
```

3.4.2 R_usb_pcdc_ReceiveData

Issue a data receive request to USB driver

Format

void	R_usb_pcdc_ReceiveData (uint8_t *table, uint32_t size, USB_CB_t complete)

Argument

*table	Pointer to transmmit data buffer address
size	Transfer size
complete	Process completion callback function

Return Value

—

Description

This function requests USB data reception of the HCD.
When the data of specified size is received or the data of less than max packet size is received, callback function is called.
The received data is stored in the area that is specified address.

Note

The USB transmit process results are found in the call-back function’s arguments.
See "USB Communication Structure" (USB_UTR_t) in the USB-BASIC-FW application note.

Example

```
void usb_smp_task( void )
{
  uint8_t    receive_data[64];                    /* Data buff */
  uint32_t   size = 64;                           /* Data size */

  R_usb_pcdc_ReceiveData(receive_data, size, &usb_complete);
}

/* Callback function */
void usb_complete(USB_UTR_t *mess)
{
  /* Processing at the time of the completion of USB reception */
}
```

3.4.3 R_usb_pcdc_SerialStateNotification

Transmit “SerialState” class notification

Format

void R_usb_pcdc_SerialStateNotification (USB_SCI_SerialState_t serial_state, USB_UTR_CB_t complete)

Argument

serial_state	Serial status
complete	Process completion callback function

Return Value

—

Description

The CDC class notification “SerialState” is transmitted to the USB host.
This transmission uses “interrupt-IN transfer”.
When the transmission is done, the callback function ‘complete’ is called.

Note

USB_SCI_SerialState_t has defined according to "Table 3.9 UART state bitmap format".
The USB transmit process results are found in the callback function’s arguments.
See "USB Communication Structure" (USB_UTR_t) in the USB-BASIC-FW application note.

Example

```
void usb_apl_task( void )
{
  USB_SCI_SerialState_t  serial_state;  /* Serial state */

  serial_state.WORD = 0x0000;
  serial_state.bParity = 0x0020;        /* D5 : Parity error */
  R_usb_pcdc_SerialStateNotification(serial_state, &usb_complete);
}

/* Callback function */
void usb_complete(USB_UTR_t *mess)
{
  /* Processing at the time of the completion of serial State transmitting */
}
```

3.4.4 R_usb_pcdc_ctrltrans

Control transfer processing for CDC

Format

void R_usb_pcdc_ctrltrans (USB_REQUEST_t *preq, uint16_t ctsq)

Argument

*preq	Pointer to a class request message		
ctsq	Control transfer stage information		
		USB_CS_IDST	Idle or setup stage
		USB_CS_RDDS	Control read data stage
		USB_CS_WRDS	Control write data stage
		USB_CS_WRND	Control write no data status stage
		USB_CS_RDSS	Control read status stage
		USB_CS_WRSS	Control write status stage
		USB_CS_SQER	Sequence error

Return Value

—

Description

Register this API to the member “*ctrltrans*” in USB_PCDREG_t structure as the callback function.
When the request type is a CDC class request, this function calls the processing that corresponds to the control transmit stage.

Note

—

Example

```
void pcdc_init(void)
{
  USB_PCDREG_t  driver;

  driver.ctrltrans = &R_usb_pcdc_ctrltrans;
  R_usb_pstd_DriverRegistration(&driver);
}
```

4. Sample Application

This section describes the initial settings necessary for using the USB PCDC and USB-BASIC-FW in combination as a USB driver and presents an example of data transfer by means of processing by the main routine and the use of API functions.

4.1 Operating Confirmation Environment

Figure 4-1 shows an example operating environment for the USB PCDC.

Table 4.1 shows the OS environment in which the operation was confirmed.

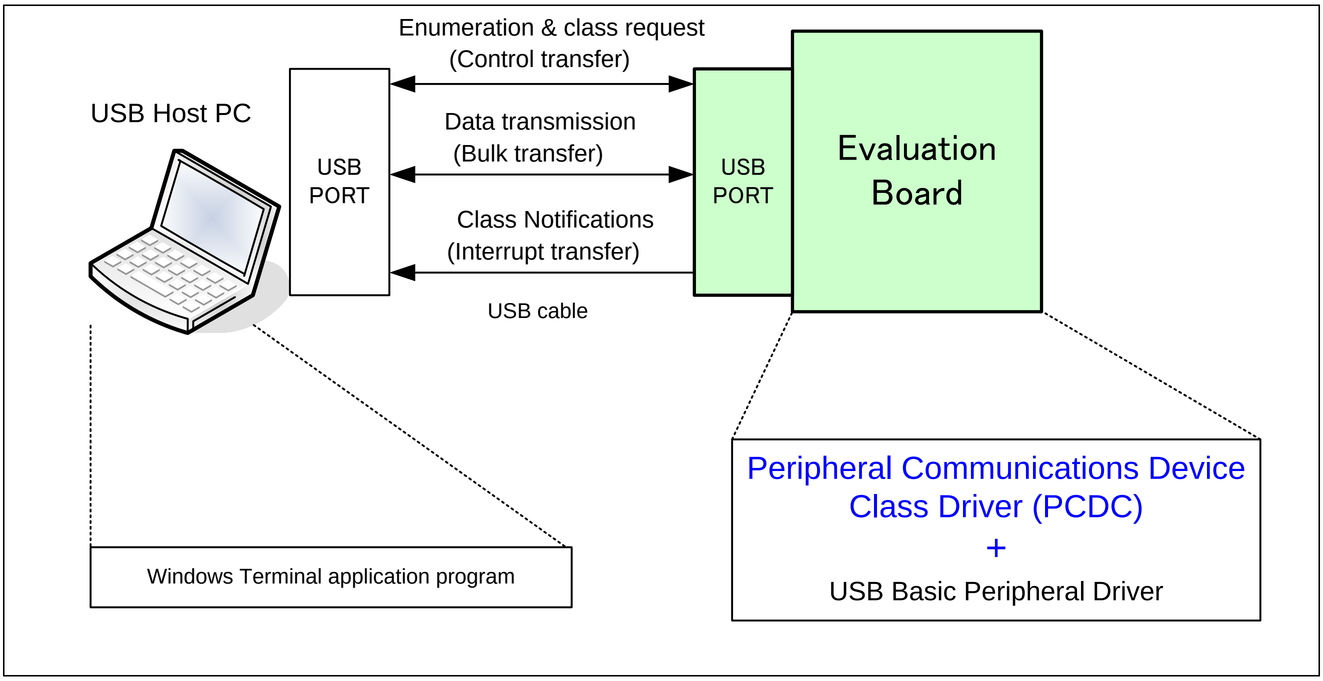


Figure 4-1 Example Operating Environment

Table 4.1 Operation Confirmed OS

OS Name	Remarks
Windows 7 32bit	—
Windows 7 64bit	—
Windows 8.1 32bit	—
Windows 8.1 64bit	—
Windows 10 32bit	—
Windows 10 64bit	—

4.2 Application Specifications

The main functions of the sample application are as follows:

1. make the initial setting of USB
2. respond to the CDC class request
3. receive the data from the USB host.
4. send the received data to the USB host

4.3 Initial Settings

Sample settings are shown below.

```
void usbf_main(void)
{
    /* USB driver setting (Refer to "4.3.1") */
     pcdc_registration();
    /* USB module initial setting (Refer to "4.3.2" */
     R_USB_Open();

     apl_init();
    /* Main routine (Refer to "4.4") */
     pcdc_main();
}
```

4.3.1 Initial setting of USB Driver

The USB driver settings consist of registering class driver information for the USB-BASIC-F/W. Refer to the USB-BASIC-FW application note "Register Peripheral Class Driver ".

4.3.2 Startup USB Module

Set the USB module according to the initial setting sequence of the hardware manual, the USB interrupt handler registration and USB interrupt enable setting.

4.4 Processing by Main Routine

After the USB driver initial settings, call the USB interrupt processing function (R_usb_pstd_poll()) from the main routine of the application. R_usb_pstd_poll() is to be called in the main routine, and the presence of interrupt is confirmed, and when interrupt occurred, it's processed according to the interrupt.

```
void pcdc_main(void)
{
    while(1)
    {
        R_usb_pstd_poll();

        switch( cdc_dev_info.state )
        {
            case STATE_DATA_TRANSFER:
                cdc_data_transfer();
                break;
            case STATE_ATTACH:
                cdc_connect_wait();
                break;
            case STATE_DETACH:
                cdc_detach_device();
                break;

            default:
                break;
        }
    }
}
```

4.4.1 APL

APL performs the following processing:

1. The APL manages the states and the events associated with them. First, the APL checks the state (see Table 4.2). It then stores the state as a member of a structure (see 4.4.2) managed by the APL.
2. Next, the APL confirms the event (see Table 4.3) associated with the state and performs the processing corresponding to the event. After event processing, the APL changes the state if necessary. Note that the event is stored as the member of a structure (see 4.4.2) managed by the APL.

An overview of the processing performed by the APL is shown below:

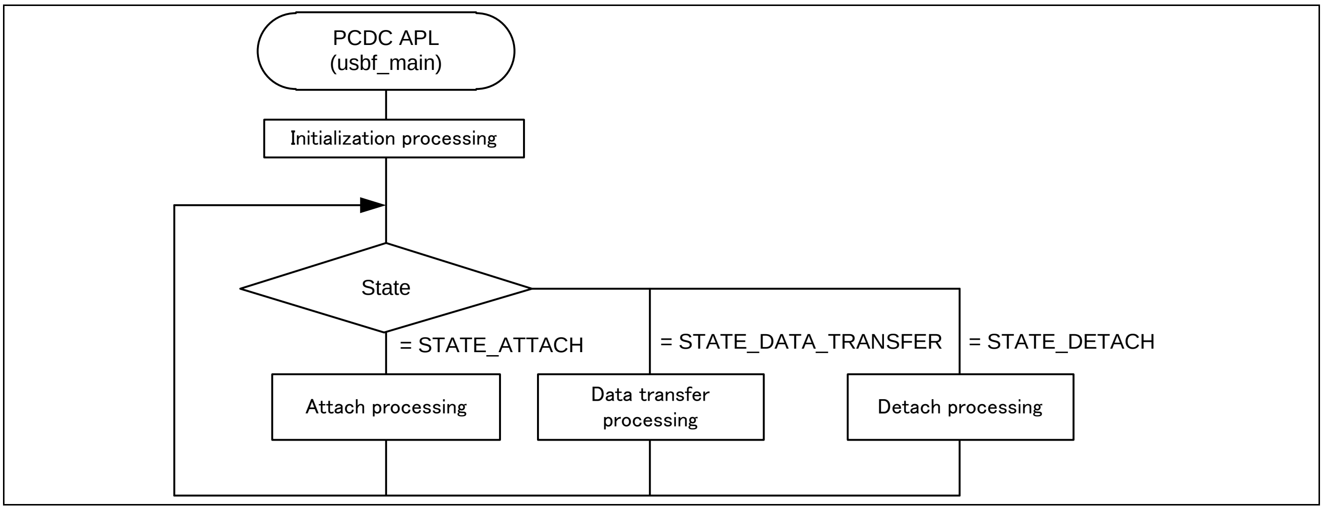


Figure 4-2 Main Loop processing

Table 4.2 List of State

State	State Processing Overview	Releated Event
STATE_ATTACH	Attach processing	EVENT_CONFIGURD
STATE_DATA_TRANSFER	Data transfer processing	EVENT_USB_READ_START
		EVENT_USB_READ_COMPLETE
		EVENT_USB_WRITE_START
		EVENT_USB_WRITE_COMPLETE
		EVENT_COM_NOTIFY_START
		EVENT_COM_NOTIFY_COMPLETE
STATE_DETACH	Detach processing	--

Table 4.3 List of Event

Event	Outline
EVENT_CONFIGURD	USB device connecting completion
EVENT_USB_READ_START	USB data reception request
EVENT_USB_READ_COMPLETE	USB data reception comletion
EVENT_USB_WRITE_START	USB data transmission request
EVENT_USB_WRITE_COMPLETE	USB data transmission completion
EVENT_COM_NOTIFY_START	Class notification “SerialState” transmission request
EVENT_COM_NOTIFY_COMPLETE	Class notification”SerialState” transmission completion
EVENT_NONE	No event

4.4.2 State and Event Management

The following structure are used to manage states and events. This structure is prepared by the APL.

If the processing to get the event determines that the event to be fetched is not present, the event is set to “EVENT_NONE.”

```
typedef struct DEV_INFO                   /* Structure for CDC device control */
{
    uint16_t    state;                    /* State for application */
    uint16_t    event_cnt;                /* Event count */
    uint16_t    event[EVENT_MAX];         /* Event. */
}
DEV_INFO_t;
```

An overview of the processing associated with each state is provided below.

1) Attach processing (STATE_ATTACH)

== Outline ==

In this state, processing is performed to notify the APL that the CDC device has connected to the USB host and enumeration has finished, after which the state changes to STATE_DATA_TRANSFER.

== Description ==

① In the APL, first the initialization function sets the state to STATE_ATTACH and the event to EVENT_NONE.

② The state continues to be STATE_ATTACH until an CDC device is connected, and *cdc_connect_wait()* is called. When a CDC device is connected and enumeration completes, the callback function *cdc_configured()* is called by the USB driver, this callback function issues the event EVENT_CONFIGURD. Note that this callback function is specified in the member *devconfig* of structure *USB_PCDREG_t*.

③ In event EVENT_CONFIGURD, the state changes to STATE_DATA_TRANSFER and the event EVENT_USB_READ_START is issued.

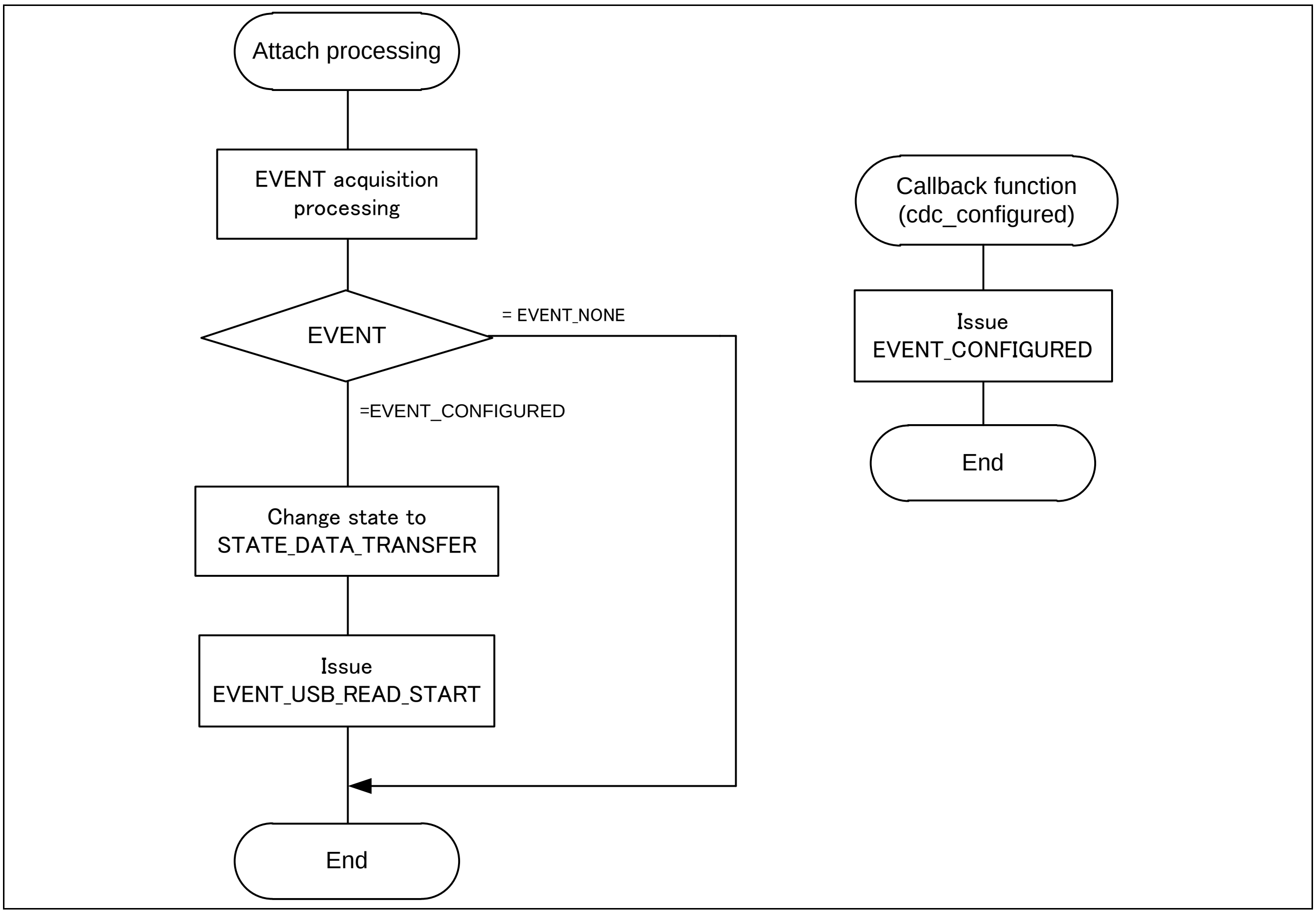


Figure 4-3 Flowchart of Attach Processing

2) Data Transfer processing (STATE_DATA_TRANSFER)

== Outline ==

In this state, until the first data is received, an initial connection message is transmitted to the USB host at intervals. After the first data received, loopback processing is performed in which data is received from the USB host, and the received data is then transmitted back to the USB host without being changed.

== Description ==

[Initial connection message]

1. When the CDC device connects to the USB host, the following message is sent to the USB host at intervals.

   PCDC.Virtual serial COM port. Type characters into terminal.
   The target will receive the characters over USB CDC, then copy them to a USB transmit buffer to be echoed back over USB."

2. When a data is sent from the terminal software running on the PC, “Echo Mode” is displayed in the terminal window. After “Echo Mode” is displayed, the CDC device runs loopback processing.

[Reception]

① EVENT_USB_READ_START sends a data receive request to the USB driver to enable reception of data transmitted by the USB host.

② When data reception finishes, the callback function *cdc_read_complete()* is called. This callback function issues EVENT_USB_READ_COMPLETE.

③ EVENT_USB_READ_COMLETE issues the event EVENT_USB_WRITE_START.

[Transmission]

④ EVENT_USB_WRITE_START sends a data transmit request to the USB driver to enable transmission to the USB host of the data received as described above.

⑤ When data transmission finishes, the callback function *cdc_write_complete()* is called. This callback function issues EVENT_USB_WRITE_COMPLETE.

⑥ EVENT_USB_WRITE_COMLETE issues the event EVENT_USB_READ_START. This causes the processing described in ① to occur again.

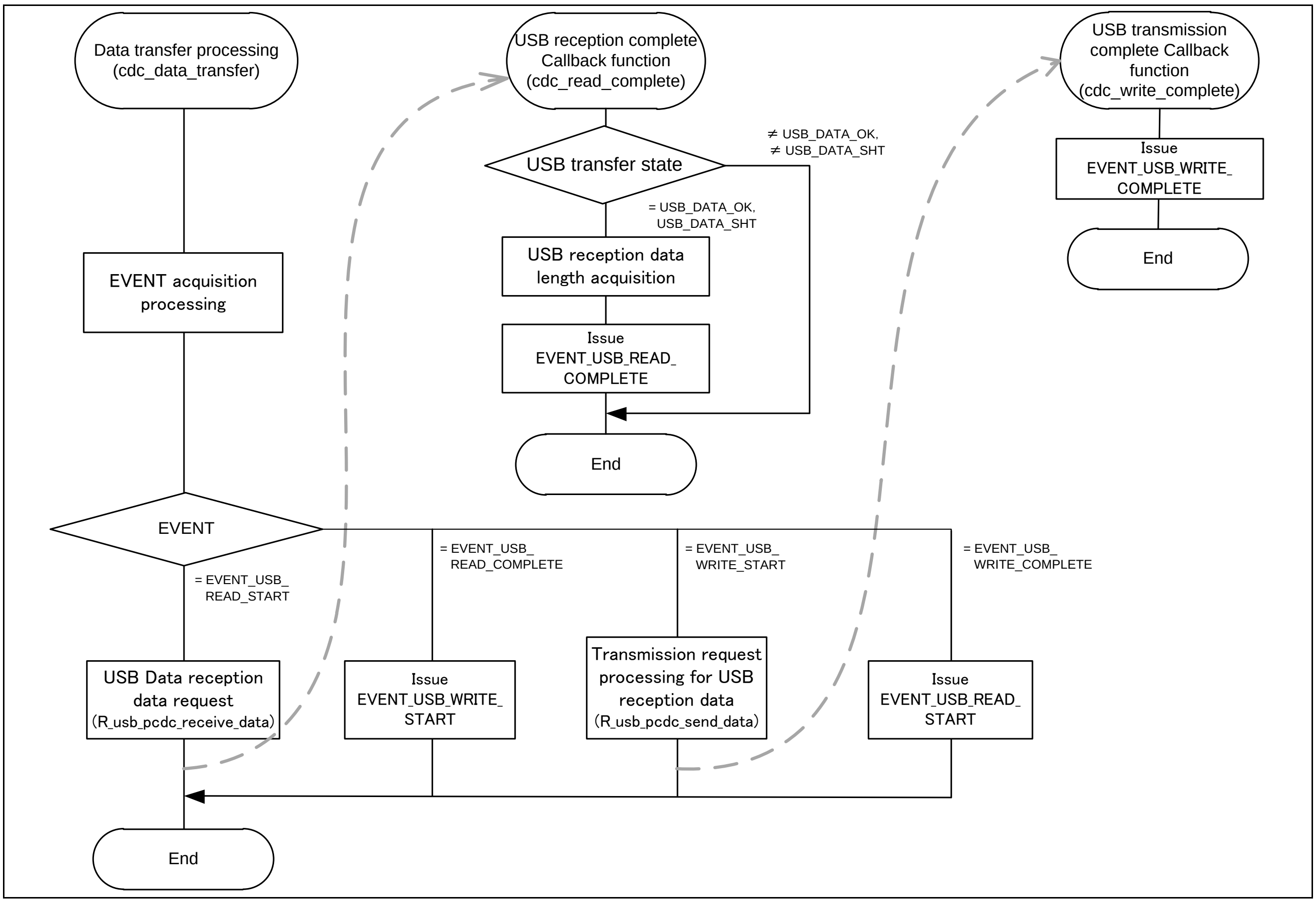


Figure 4-4 Flowchart of Data Transfer Processing

3) Detach processing (STATE_DETACH)

When the CDC device disconnect from the connected USB Host, the USB driver calls the callback function *cdc_detach()*. This callback function changes the state to STATE_DETACH. In STATE_DETACH, processing is performed to clear variables and change the state to STATE_ATTACH, among other things.

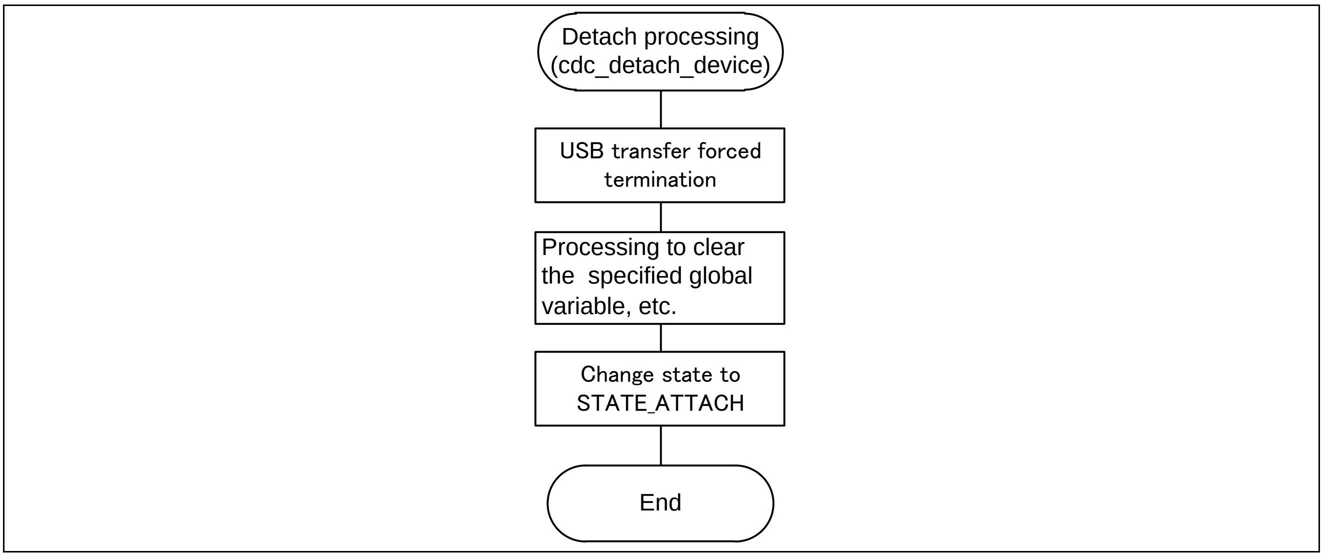


Figure 4-5 Flowchart of Detach Processing

Appendix A. Changes of initial setting

USB-BASIC-F/W has been changed to "RZ/T1 group initial setting Rev.1.30".
Sample program supports IAR embedded workbench for ARM (EWARM) ,DS-5 and e^2 studio.
This chapter describes the changes.

Folders and files

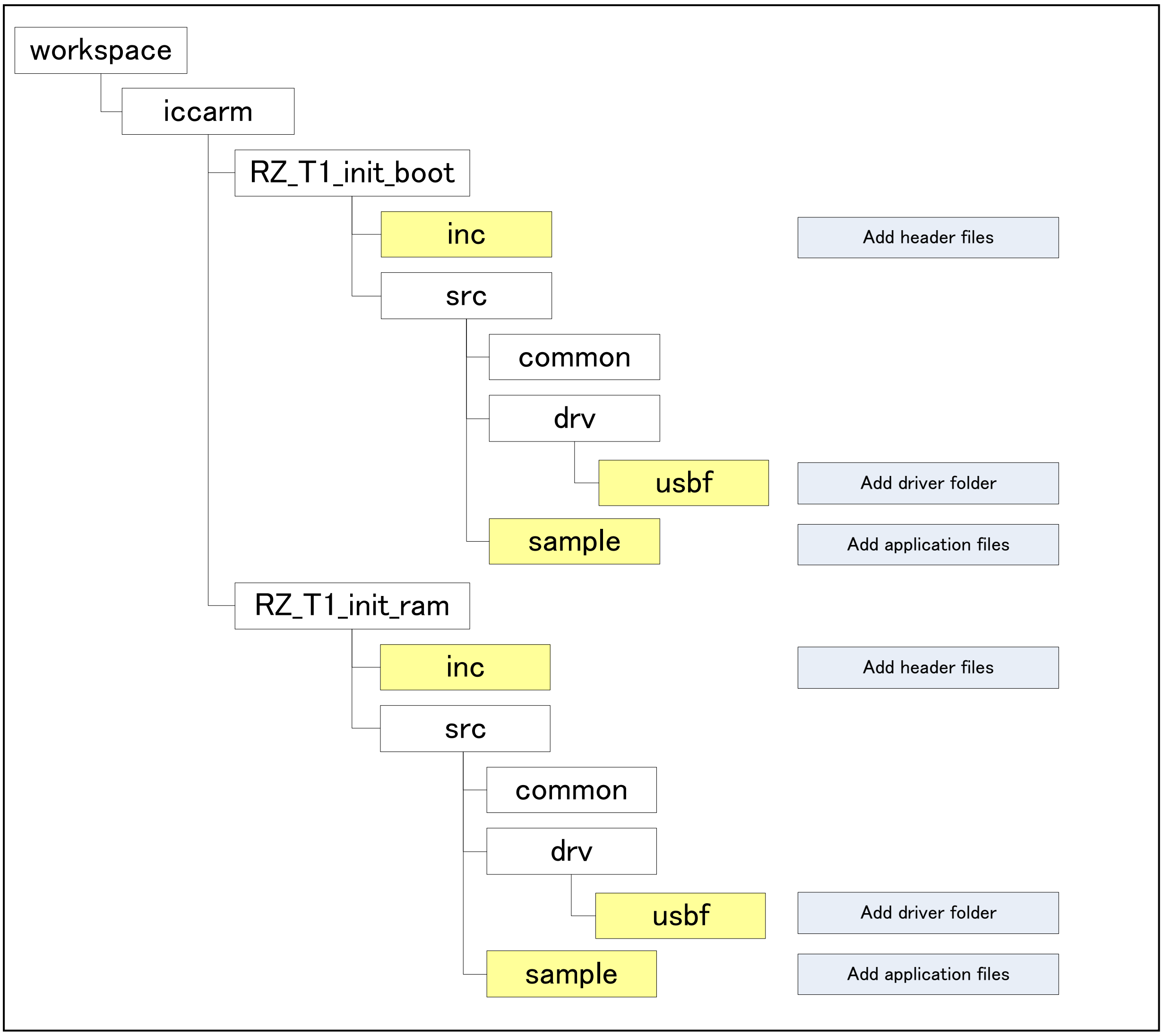
In the "RZ/T1 group initial setting Rev.1.30", different folder structure by the development environment and the boot method. Changes to each folder of all of the development environment and the boot method it is shown below.

- Add the following files in the “inc” folder.
  r_usb_basic_config.h
  r_usb_basic_if.h
  r_usb_cdefusbip.h
  r_usb_pcdc_config.h
  r_usb_pcdc_if.h

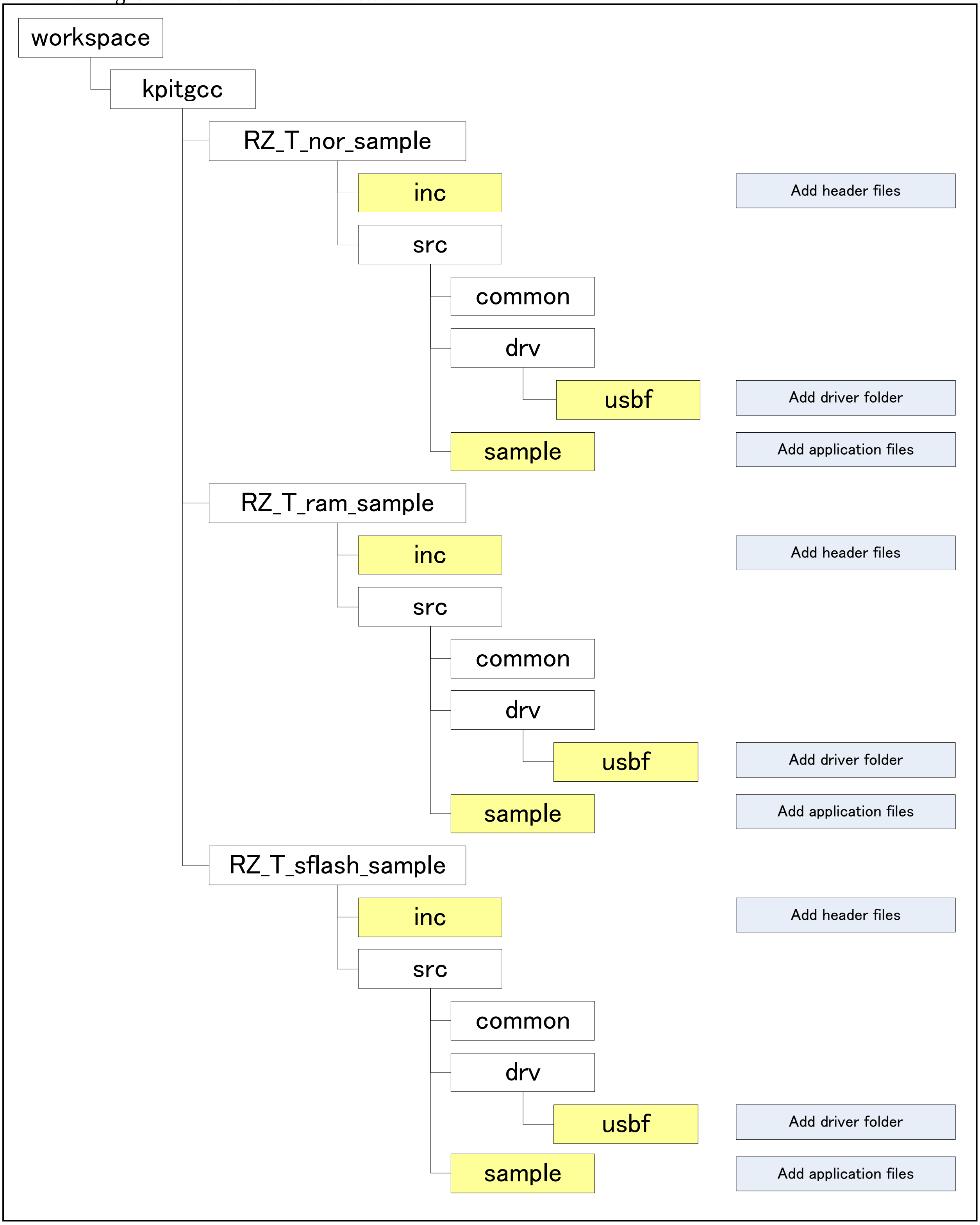
- Add the following files in the “sample” folder.
  r_usb_pcdc_apl.c
  r_usb_pcdc_descriptor.c

- Add the “usbf” folder and the following files “usbf” folder in the “drv” folder.

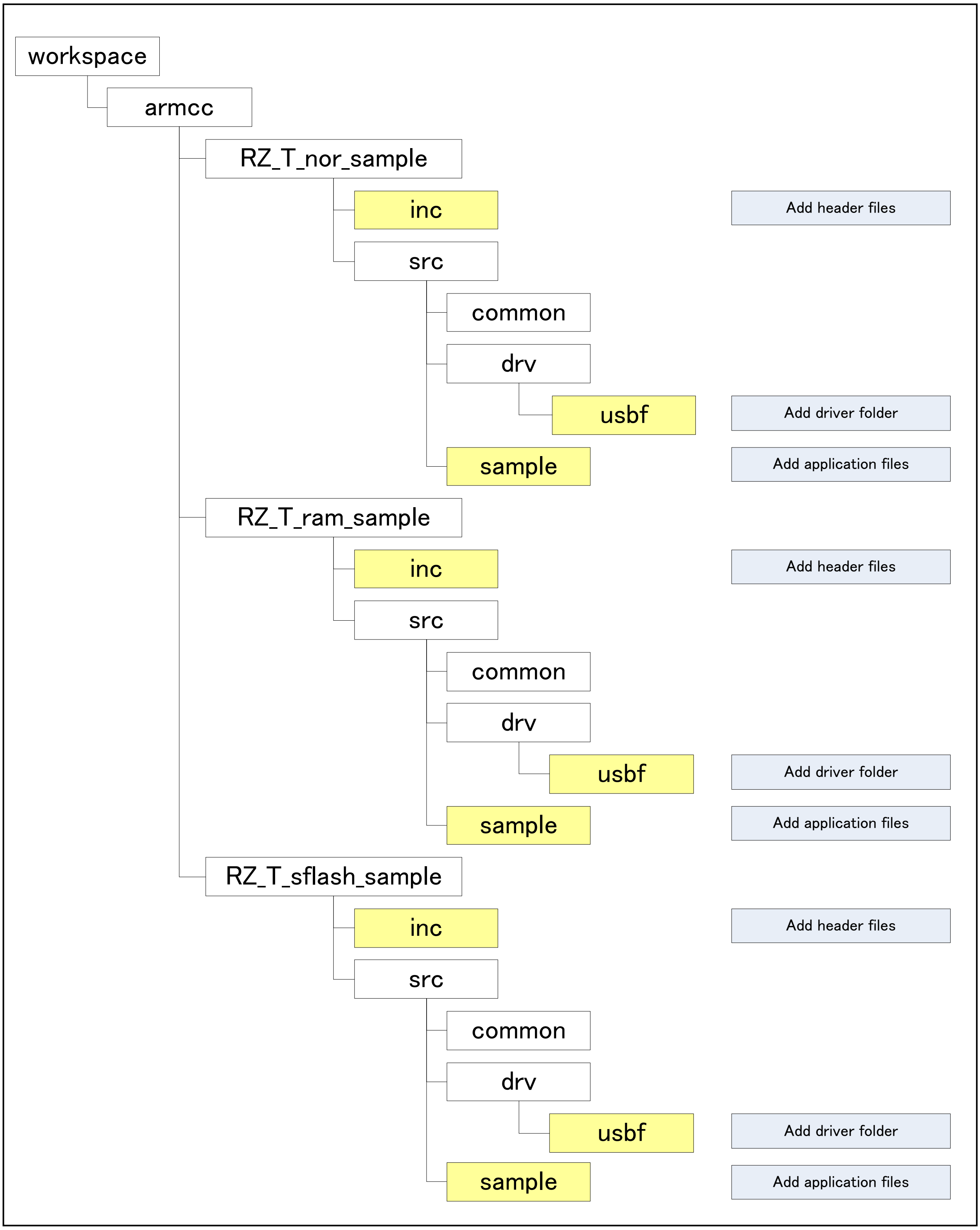
The following is the folder structure of EWARM.



The following is the folder structure of e[2] studio.



The following is the folder structure of DS-5.



Call the USB-BASIC-FW function

Adds the usbf_main() of USB-BASIC-F/W in the main() of "\src\sample\int_main.c".

```
extern void usbf_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBf main */
    usbf_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug 21, 2015	—	First edition issued
1.10	Dec 25, 2015	19	Added Appendix A
1.20	Feb 29, 2016	21	Added DS-5 setting
1.30	Dec 07, 2017	—	Corresponds to RZ / T1 initial setting Ver 1.30

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance 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 a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0