User's Manual

CONFIDENTIAL

Linux Interface Specification Device Driver USB 3.0 Function

User's Manual: Software

R-Car H3/M3/M3N/E3 Series

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.3.00 Dec. 2021

Notice

1. 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

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

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall 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. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Trademark

- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- Other company names and product names mentioned herein are registered trademarks or trademarks of their respective owners.
- Registered trademark and trademark symbols (® and ™) are omitted in this document

How to Use This Manual

- **[Readers]**
  This manual is intended for engineers who develop products which use the R-Car H3/M3/M3N/E3 processor.

- **[Purpose]**
  This manual is intended to give users an understanding of the functions of the R-Car H3/M3/M3N/E3 processor device driver and to serve as a reference for developing hardware and software for systems that use this driver.

- **[How to Read This Manual]**
  It is assumed that the readers of this manual have general knowledge in the fields of electrical
  — engineering, logic circuits, microcontrollers, and Linux.
  → Read this manual in the order of the CONTENTS.
  — To understand the functions of a multimedia processor for R-Car H3/M3/M3N/E3
  → See the R-Car H3/M3/M3N/E3 User's Manual.
  — To know the electrical specifications of the multimedia processor for R-Car H3/M3/M3N/E3
  → See the R-Car H3/M3/M3N/E3 Data Sheet.

- **[Conventions]**
  The following symbols are used in this manual.
  Data significance: Higher digits on the left and lower digits on the right
  Note: Footnote for item marked with Note in the text
  Caution: Information requiring particular attention
  Remark: Supplementary information
  Numeric representation: Binary ... ××××, 0b××××, or ××××B
  Decimal ... ××××
  Hexadecimal ... 0x×××× or ××××H
  Data type: Double word … 64 bits
  Word … 32 bits
  Half word ... 16 bits
  Byte ... 8 bits

Table of Contents

1. Overview

1.1 Overview

This manual explains the driver module (this module) that controls the USB 3.0 Function controller on R-Car H3/M3/M3N/E3.

1.2 Function

This module controls USB 3.0 Function controller on R-Car H3/M3/M3N/E3, and transmission and reception of data are performed by USB3.0 standard between USB Host connected to the USB interface.

This module supports only role swap don’t using Host Negotiation Protocol (HNP). This module role swap to USB Host. No support for Session Request Protocol (SRP).

Supports USB2.0 Speed types, High and Full Speed.

The maximum PIPE is 5 except PIPE0 and EP0.

Able to be set to the following configurations for each PIPE with USB3.0 Super Speed.

- Endpoint Number: 0 to 5
- Transfer Type: Control(PIPE0 only)/Bulk/Interrupt
- Max Burst Size: 1(value fixed. PIPE0 Control), 1 to 16(Bulk), 1 to 3(Interrupt)
- Max Sequence Number:31(fixed)

Able to be set to the following configurations for each PIPE with USB2.0 Speed.

- Endpoint Number:0 to 5
- Transfer Type: Control(PIPE0 only)/Bulk/Interrupt

1.3 Connected Port

This module supports one USB ports on R-CarH3-SiP/M3-SiP/M3N/E3 System Evaluation Board.

Table 1-1 Connected Port

Port No.	Standard	Connector No.	Content
0	USB3.0 Host/Function	CN11	Type A connector

1.4 Reference Document

1.4.1 Standard

Supported standard of this module is as follows.

Table 1-2 Standard

Reference No.	Issue	Title	Edition	Date
-	USB Implementers Forum, Inc	Universal Serial Bus 3.0 Specification	Rev.1.0	Jun. 6, 2011

1.4.2 Related Document

Related document of this module is as follows.

Table 1-3 Related Documents

Reference No.	Issue	Title	Edition	Date
-	Renesas Electronics	R-Car Series, 3rd Generation User's Manual: Hardware	Rev.2.20	Jun. 30, 2020
-	Renesas Electronics	R-CarH3-SiP/M3-SiP/M3N-SiP System Evaluation Board Salvator-XS Hardware Manual	Rev.2.04	Jul. 17, 2018
-	Renesas Electronics	R-CarH3-Sip System Evaluation Board Salvator-X Hardware Manual RTP0RC7795SIPB0011S	Rev.1.09	May. 11, 2017
-	Renesas Electronics	R-CarM3-SiP System Evaluation Board Salvator-X Hardware Manual RTP0RC7796SIPB0011S	Rev.0.04	Oct. 3, 2016
-	Renesas Electronics	R-CarE3 System Evaluation Board Ebisu Hardware Manual RTP0RC77990SEB0010S	Rev.0.03	Apr. 11, 2018
-	Renesas Electronics	R-CarE3 System Evaluation Board Ebisu-4D (E3 board 4xDRAM) Hardware Manual	Rev.1.01	Jul. 19, 2018

1.5 Restrictions

There is no restriction in this module.

1.6 Notice

The notes of this module are shown below.

- This module supports H3 (Ver.2.0 or later), M3 (Ver.1.1 or later) and M3N (Ver.1.1 or later) and E3.
- This module implements switching by ID pin in role swap function. But the function is unconfirmed because Evaluation Board does not implement ID pin. Please use USB 2.0 cable, because role swap is supported with High-Speed only.
- The USB3.0 CN11 connector of Salvator-X, Salvator-XS and Ebisu board is for USB Host. Please do not connect CN11 connector and USB Host device.
- The known problem in the standard gadget class driver for Linux is not supported.
- Only the standard Gadget interface for Linux is supported by this module.

2. Terminology

The following table shows the terminology related to this module.

Table 2.1 Terminology

Terms	Explanation
USB	USB Universal Serial Bus
UDC	USB Device Controller
EP	Endpoint
OTG	On-The-GO
HNP	Host Negotiation Protocol
SRP	Session Request Protocol

3. Operating Environment

3.1 Hardware Environment

The following table lists the hardware needed to use this module.

Table 3.1 Hardware specification

Name	Version	Manufacture
R-CarH3-Sip System Evaluation Board Salvator-X	-	Renesas Electronics
R-CarM3-Sip System Evaluation Board Salvator-X	-	Renesas Electronics
R-CarH3-SiP/M3-SiP/M3N-SiP System Evaluation Board Salvator-XS	-	Renesas Electronics
R-CarE3 System Evaluation Board Ebisu	-	Renesas Electronics
R-CarE3 System Evaluation Board Ebisu-4D	-	Renesas Electronics

3.2 Module Configuration

The following figure shows the configuration of this module.

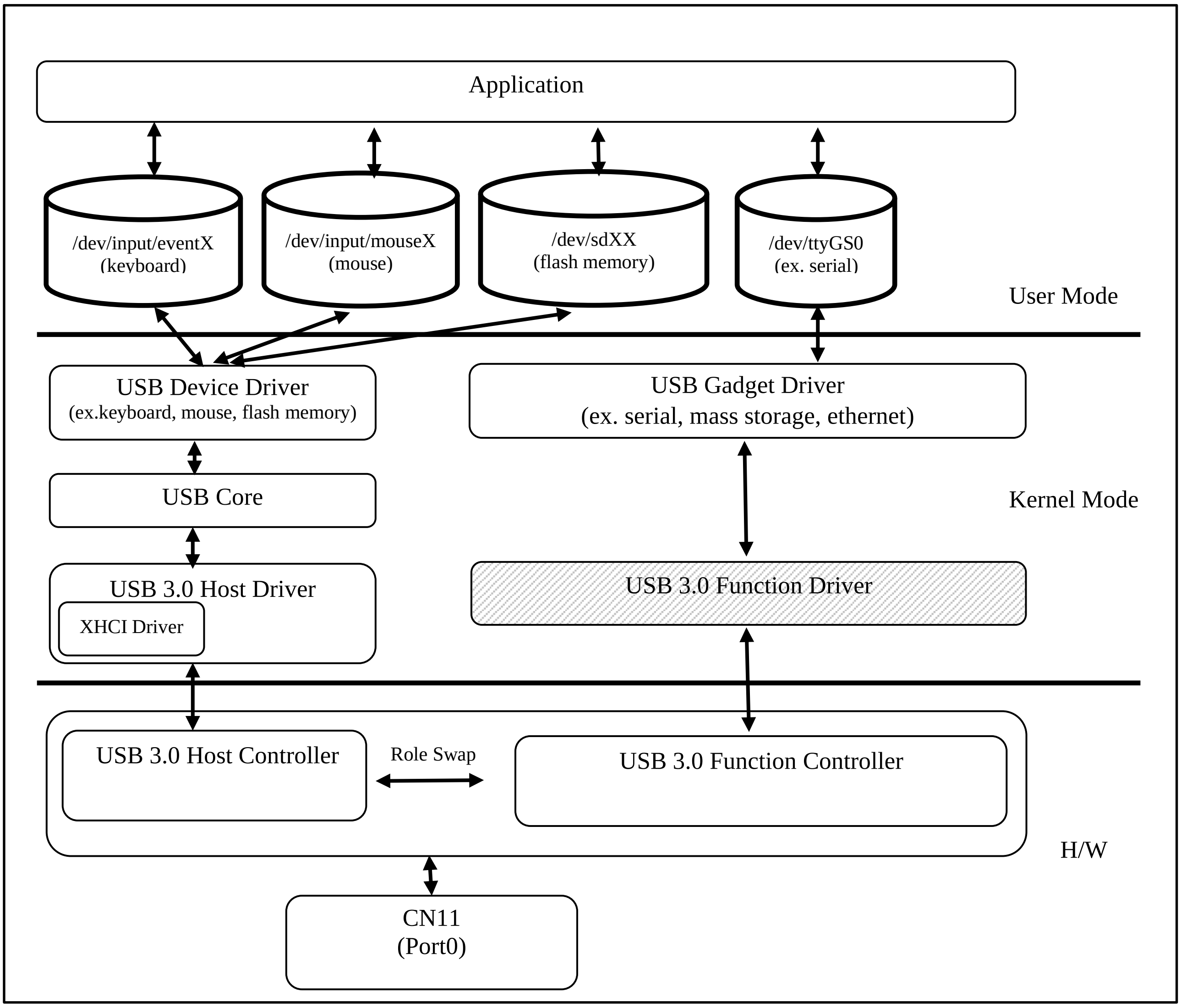

Figure 3-1 Module Configuration

3.3 State Transition Diagram

There is no state transition diagram for this module.

4. External Interface

The supported external interface of this module is explained.

Since it is based on USB standard, the definition described in "include/linux/usb/ch9.h" is omitted.

4.1 Function specification

The interface function list which this module supports is shown below.

When you use these functions, please include the following header file.

```
#include <linux/usb/gadget.h>
```

Table 4.1 List of interface function (register / unregister)

Chapter	Function name	Remarks
4.1.1	usb_gadget_probe_driver	A Gadget driver is registered to UDC
4.1.2	usb_gadget_unregister_driver	A Gadget driver is released from UDC.

Table 4.2 List of interface function (endpoint-specific operations)

Chapter	Function name	Remarks
4.1.3	usb_ep_enable	Configure endpoint, making it usable
4.1.4	usb_ep_disable	endpoint is no longer usable
4.1.5	usb_ep_alloc_request	Allocate a request object to use with this endpoint
4.1.6	usb_ep_free_request	Free a request object
4.1.7	usb_ep_queue	Queue (submit) an I/O request to an endpoint
4.1.8	usb_ep_dequeue	Dequeue (cancel, unlink) an I/O request from an endpoint
4.1.9	usb_ep_set_halt	Endpoint is changed into a STALL state
4.1.10	usb_ep_clear_halt[*1]	STALL of Endpoint is canceled.
4.1.11	usb_ep_set_wedge	Endpoint is changed into a STALL state
4.1.12	usb_ep_fifo_status[*1]	Return number of bytes in FIFO, or error
4.1.13	usb_ep_fifo_flush	Flush contents of a FIFO

Notes:1. This interface is unsupported.

Table 4.3 List of interface function (hardware-specific operations)

Chapter	Function name	Remarks
4.1.14	gadget_is_dualspeed	Return true if the hardware handles high speed
4.1.15	gadget_is_superspeed	Return true if the hardware is super speed ready
4.1.16	gadget_is_otg	Return true if the hardware is OTG-ready
4.1.17	usb_gadget_frame_number	Return the current frame number
4.1.18	usb_gadget_wakeup*1	Try to wake up the host connected to this gadget
4.1.19	usb_gadget_set_selfpowered	Set the device self-powered feature
4.1.20	usb_gadget_clear_selfpowered*1	Clear the device self-powered feature
4.1.21	usb_gadget_vbus_connect*1	Supply power to VBUS
4.1.22	usb_gadget_vbus_draw*1	Notify VBUS power usage
4.1.23	usb_gadget_vbus_disconnect*1	Cancel power to VBUS
4.1.24	usb_gadget_connect	Turn on D+ Pull-up
4.1.25	usb_gadget_disconnect	Turn off D+ Pull-up

Notes:1. This interface is unsupported.

From the following chapter, the function which this module supports is explained according to the following description formats.

[Overview]	Presents an overview of a function.
[Function Name]	Explains the name of the function.
[Calling format]	Explains the format for calling the function.
[Argument]	Explains the argument(s) of the function.
[Return value]	Explains the return value(s) of the function.
[Feature]	Explains the features of the function.
[Remark]	Explains points to be noted when using the function.

4.1.1 usb_gadget_probe_driver

[Overview] A Gadget driver is registered to UDC

[Function Name] usb_gadget_probe_driver

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_gadget_probe_driver(
struct usb_gadget_driver *driver,
int (*bind)(strunct usb_gadget*));
```

[Argument]

driver	: Pointer of USB Gadget driver structure
bind	: Callback function pointer at the time of USB Gadget driver registration

[Return value]

0	: Normal termination
-ENODEV	: UDC is not registered
-EBUSY	: Gadget is already registered into UDC
-EINVAL	: Invalid argument
less than 0	: other error

[Feature] A Gadget driver is registered to UDC by setting up a usb_gadget_driver structure and calling this function.
The usb_gadget_driver structure can register only one to UDC.

[Remark] Please refer to 4.2.6 about usb_gadget_driver structure
Please perform acquisition of a workspace memory, registration of driver data, maintenance of a usb_gadget structure, an automatic setup of Endpoint Descriptor, etc. in a callback function at the time of registration.
It cannot call out of an interrupt handler.

4.1.2 usb_gadget_unregister_driver

[Overview] A Gadget driver is released from UDC

[Function Name] usb_gadget_unregister_driver

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_gadget_unregister_driver(
struct usb_gadget_driver *driver);
```

[Argument]

driver	: Pointer of USB Gadget driver structure

[Return value]

0	: Normal termination
-ENODEV	: UDC is not registered.
-EINVAL	: Invalid argument

[Feature] By setting the usb_gadget_driver structure of the same pointer as the time of registering as an argument, and calling this function, a Gadget driver is released from UDC.

[Remark] Please refer to 4.2.6 about usb_gadget_driver structure
Please perform release of a workspace memory, deletion of driver data, deletion of a usb_gadget_driver structure, etc. in a callback function at the time of registration release.
It cannot call out of an interrupt handler.

4.1.3 usb_ep_enable

[Overview] Endpoint is enabled

[Function Name] usb_ep_enable

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_ep_enable(
struct usb_ep *ep,
const struct usb_endpoint_descriptor *desc);
```

[Argument]

ep	: Pointer of USB Endpoint structure
desc	: Pointer Endpoint Descriptor

[Return value]

0	: Normal termination
-ENODEV	: UDC is not registered.
-EINVAL	: Invalid argument
-ENOMEM	: Out of memory

[Feature] It sets up for enabling Endpoint

[Remark] Please refer to 4.2.4 about usb_ep structure
Please enable Endpoint with this function after acquiring a usb_ep structure required for Endpoint processing.
It can call from an interrupt handler.

4.1.4 usb_ep_disable

[Overview] Endpoint is disabled

[Function Name] usb_ep_disable

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_ep_disable(
struct usb_ep *ep);
```

[Argument]

ep	: Pointer of USB Endpoint structure

[Return value]

0	: Normal termination

[Feature] It sets up for disabling Endpoint

[Remark] Please refer to 4.2.4 about usb_ep structure
Any pending and uncompleted requests will complete with status indicating disconnect (-ESHUTDOWN) before this call returns.
As for Endpoint under transmission, transmission is stopped.
In order not to generate a memory leak, a usb_ep_disable() front or status is within the complete call-back of -ESHUTDOWN, Please be sure to release a usb_request structure by usb_ep_free_request().
It can call from an interrupt handler.

4.1.5 usb_ep_alloc_request

[Overview] Allocate a request object to use with this endpoint

[Function Name] usb_ep_alloc_request

[Calling format]

```
#include <linux/usb/gadget.h>
struct usb_request *usb_ep_alloc_request(
struct usb_ep *ep,
gfp_t gfp_flags);
```

[Argument]
ep : Pointer of USB Endpoint structure
gfp_flags : flag of memory acquisition (GFP_KERNEL / GFP_ATOMIC)

[Return value]
Except 0 : Normal termination
0 : Failed to allocate memory for a request object

[Feature] The information structure object (USB Endpoint transmission request structure) for performing the transmission and reception to Endpoint is acquired.

[Remark]
Please refer to 4.2.4 about usb_ep structure
Please be sure to acquire using this function, when creating a usb_request structure.
Please be sure to perform usb_ep_free_request and to release a memory after the completion of use.
By setting up a memory acquisition flag appropriately, it can call from an interrupt handler.

4.1.6 usb_ep_free_request

[Overview] A USB Endpoint transmission request structure is released.

[Function Name] usb_ep_free_request

[Calling format]

```
#include <linux/usb/gadget.h>
void usb_ep_free_request(
struct usb_ep *ep,
struct usb_request *req);
```

[Argument]
ep : Pointer of USB Endpoint structure
req : Pointer USB Endpoint transmission request structure

[Return value] none

[Feature] The usb_request structure acquired by usb_ep_alloc_request() is released.

[Remark]
Please refer to 4.2.4 about usb_ep structure
Please refer to 4.2.1about usb_request structure
Be careful of whether buffer in a usb_request structure is released before release.
It can call from an interrupt handler.

4.1.7 usb_ep_queue

[Overview] USB Endpoint transmission is required.

[Function Name] usb_ep_queue

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_ep_queue(
struct usb_ep *ep,
struct usb_request *req
gfp_t gfp_flags);
```

[Argument]

ep	: Pointer of USB Endpoint structure
req	: Pointer USB Endpoint transmission request structure
gfp_flags	: flag of memory acquisition

[Return value]

0	: Normal termination
-ESHUTDOWN	: The speed of USB of operation is un-setting up.

[Feature] A data transfer setup is performed according to a setup of usb_request structure.

[Remark] Please refer to 4.2.4 about usb_ep structure
Please refer to 4.2.1about usb_request structure
By setting up a memory acquisition flag appropriately, it can call from an interrupt handler.

4.1.8 usb_ep_dequeue

[Overview] USB Endpoint transmission is stopped.

[Function Name] usb_ep_dequeue

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_ep_dequeue(
struct usb_ep *ep,
struct usb_request *req);
```

[Argument]

ep	: Pointer of USB Endpoint structure
req	: Pointer USB Endpoint transmission request structure

[Return value]

0	: Normal termination

[Feature] A transmission stop will be carried out, if the specified usb_request structure is in transmission queue or it is under transmission.（-ECONNRESET is put into the status member of usb_request and completion call-back is carried out.）

[Remark] Please refer to 4.2.4 about usb_ep structure
Please refer to 4.2.1about usb_request structure
It can call from an interrupt handler.

4.1.9 usb_ep_set_halt

[Overview]	Specified Endpoint is changed into a STALL state.	
[Function Name]	usb_ep_set_halt	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_ep_set_halt(
	struct usb_ep *ep);	

[Argument]	ep	: Pointer of USB Endpoint structure
[Return value]	0 -EAGAIN	: Normal termination : Double call
[Feature]	Specified Endpoint is changed into a STALL state.	
[Remark]	Please refer to 4.2.4 about usb_ep structure. It can call from an interrupt handler.	

4.1.10 usb_ep_clear_halt

[Overview]	STALL of Endpoint is canceled.	
[Function Name]	usb_ep_clear_halt	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_ep_clear_halt(
	struct usb_ep *ep);	

[Argument]	ep	: Pointer of USB Endpoint structure
[Return value]	0 -EAGAIN	: Normal termination : Double call
[Feature]	STALL of specified Endpoint is canceled.	
[Remark]	Please refer to 4.2.4 about usb_ep structure It can call from an interrupt handler.	

4.1.11 usb_ep_set_wedge

[Overview]	Endpoint is changed into a STALL state.	
[Function Name]	usb_ep_set_wedge	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_ep_set_wedge(
	struct usb_ep *ep);	

[Argument]	ep	: Pointer of USB Endpoint structure
[Return value]	0 -EAGAIN -EINVAL	: Normal termination : Double call : Invalid argument
[Feature]	Specified Endpoint is changed into a STALL state.	
[Remark]	It becomes the same processing as usb_ep_set_halt. Please refer to 4.2.4 about usb_ep structure. It can call from an interrupt handler.	

4.1.12 usb_ep_fifo_status［unsupported］

[Overview]	The data size which remains in the buffer of Endpoint is returned.	
[Function Name]	usb_ep_fifo_status	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_ep_fifo_status(
	struct usb_ep *ep);	

[Argument]	ep	: Pointer of USB Endpoint structure
[Return value]	-EOPNOTSUPP	: unsupported
[Feature]	The data size which remains in the buffer of specified Endpoint is returned.	
[Remark]	The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported Please refer to 4.2.4 about usb_ep structure. It can call from an interrupt handler.	

4.1.13 usb_ep_fifo_flush［unsupported］

[Overview]	The buffer of Endpoint is cleared.	
[Function Name]	usb_ep_fifo_flush	

[Calling format]	#include <linux/usb/gadget.h>
	void usb_ep_fifo_flush(
	struct usb_ep *ep);	

[Argument]	ep	: Pointer of USB Endpoint structure
[Return value]	none	
[Feature]	The buffer of specified Endpoint is cleared.	
[Remark]	Please refer to 4.2.4 about usb_ep structure. It can call from an interrupt handler.	

4.1.14 gadget_is_dualspeed

[Overview]	Transmission Speed which UDC supports is acquired.	
[Function Name]	gadget_is_dualspeed	

[Calling format]	#include <linux/usb/gadget.h>
	int gadget_is_dualspeed(
	struct usb_gadget *g);	

[Argument]	g	: Pointer of USB Gadget structure
[Return value]	0 1	: Full Speed or Low Speed : High Speed
[Feature]	Transmission Speed which UDC supports is acquired.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.15 gadget_is_superspeed

[Overview]	It is acquired whether UDC is supporting Super Speed.	
[Function Name]	gadget_is_superspeed	

[Calling format]	#include <linux/usb/gadget.h>
	int gadget_is_superspeed(
	struct usb_gadget *g);	

[Argument]	g	: Pointer of USB Gadget structure
[Return value]	0 1	: Super Speed is unsupported : Super Speed is supported
[Feature]	Transmission Speed which UDC supports is acquired.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.16 gadget_is_otg

[Overview]	It is acquired whether UDC is supporting OTG.	
[Function Name]	gadget_is_otg	

[Calling format]	#include <linux/usb/gadget.h>
	int gadget_is_otg(
	struct usb_gadget *g);	

[Argument]	g	: Pointer of USB Gadget structure
[Return value]	0 1	: OTG is unsupported : OTG is supported
[Feature]	It is acquired whether UDC is supporting OTG.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.17 usb_gadget_frame_number

[Overview]	Frame Number is acquired.	
[Function Name]	usb_gadget_frame_number	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_frame_number(
	struct usb_gadget *gadget):	

[Argument]	gadget	: Pointer of USB Gadget structure
[Return value]	Integer	: Frame Number
[Feature]	Current Frame Number is acquired.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.18 usb_gadget_wakeup［unsupported］

[Overview] A remote wake up is performed to USB HOST.

[Function Name] usb_gadget_wakeup

[Calling format]

```
#include <linux/usb/gadget.h>
int usb_gadget_wakeup(
struct usb_gadget *gadget);
```

[Argument] gadget : Pointer of USB Gadget structure

[Return value] -EOPNOTSUPP : Unsupported

[Feature] A remote wake up is performed to USB HOST.

[Remark] The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported.
Please refer to 4.2.6 about usb_gadget structure.
It can call from an interrupt handler.

4.1.19 usb_gadget_set_selfpowered［unsupported］

[Overview] The electric supply method of a USB device is set as self-power.

[Function Name] usb_gadget_set_selfpowered

[Calling format]

```
#include <linux/usb/gadget.h>
usb_gadget_set_selfpowered(
struct usb_gadget *gadget);
```

[Argument] gadget : Pointer of USB Gadget structure

[Return value] -EOPNOTSUPP : Unsupported

[Feature] A setup using electric power sauce other than VBUS is performed as electric power of a USB device of operation.

[Remark] The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported.
Please refer to 4.2.6 about usb_gadget structure.
It can call from an interrupt handler.

4.1.20 usb_gadget_clear_selfpowered［unsupported］

[Overview]	The electric supply method of a USB device is set as bus power.
[Function Name]	usb_gadget_clear_selfpowered

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_clear_selfpowered(
	struct usb_gadget *gadget);

[Argument]	gadget : Pointer of USB Gadget structure
[Return value]	-EOPNOTSUPP : Unsupported
[Feature]	The electric supply method of a USB device is set as bus power.
[Remark]	The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported. Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.

4.1.21 usb_gadget_vbus_connect［unsupported］

[Overview]	A power supply is supplied to VBUS.
[Function Name]	usb_gadget_vbus_connect

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_vbus_connect(
	struct usb_gadget *gadget);

[Argument]	gadget : Pointer of USB Gadget structure
[Return value]	-EOPNOTSUPP : Unsupported
[Feature]	A power supply is supplied to VBUS.
[Remark]	The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported. Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.

4.1.22 usb_gadget_vbus_draw［unsupported］

[Overview]	A current consumption value of VBUS is notified.	
[Function Name]	usb_gadget_vbus_draw	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_vbus_draw(
	struct usb_gadget *gadget,
	unsigned mA);	

[Argument]	gadget mA	: Pointer of USB Gadget structure : Consumption current value
[Return value]	0	: Normal termination
[Feature]	It calls at the time of SET_CONFIGURATION, SET_CONFIGURATION processing finishes, and it notifies the consumption current value after operation of apparatus.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.23 usb_gadget_vbus_disconnect［unsupported］

[Overview]	The electric supply of VBUS is canceled.	
[Function Name]	usb_gadget_vbus_disconnect	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_vbus_disconnect(
	struct usb_gadget *gadget);	

[Argument]	gadget	: Pointer of USB Gadget structure
[Return value]	-EOPNOTSUPP	: Unsupported
[Feature]	The electric supply of VBUS is canceled.	
[Remark]	The returned value of this function is (-EOPNOTSUPP) at a usual state for unsupported. Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.24 usb_gadget_connect

[Overview]	The pull-up resistor of D+ signal is turned ON.	
[Function Name]	usb_gadget_connect	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_connect(
	struct usb_gadget *gadget);	

[Argument]	gadget	: Pointer of USB Gadget structure
[Return value]	0	: Normal termination
[Feature]	The pull-up resistor of D+ signal is turned ON.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.1.25 usb_gadget_disconnect

[Overview]	The pull-up resistor of D+ signal is turned OFF.	
[Function Name]	usb_gadget_disconnect	

[Calling format]	#include <linux/usb/gadget.h>
	int usb_gadget_disconnect(
	struct usb_gadget *gadget);	

[Argument]	gadget	: Pointer of USB Gadget structure
[Return value]	0	: Normal termination
[Feature]	The pull-up resistor of D+ signal is turned OFF.	
[Remark]	Please refer to 4.2.6 about usb_gadget structure. It can call from an interrupt handler.	

4.2 Structure

The list of structure definitions required of this module is shown as follows.

When you use these structures, please include the following header files.

#include <linux/usb/gadget.h>

Table 4.4 List of structure

Chapter	Name of symbol in structure	Name of structure
4.2.1	usb_request	USB Endpoint transfer request structure
4.2.2	usb_ep_ops	USB Endpoint operation structure
4.2.3	usb_ep_caps	USB Endpoint capability structure
4.2.4	usb_ep	USB Endpoint structure
4.2.5	usb_gadget_ops	USB Gadget operation structure
4.2.6	usb_gadget	USB Gadget structure
4.2.7	usb_gadget_driver	USB Gadget driver structure

4.2.1 usb_request structure

```
struct usb_request {
        void                    *buf;
        unsigned                length;
        dma_addr_t              dma;
        struct scatterlist      *sg;
        unsigned                num_sgs;
        unsigned                num_mapped_sgs;
        unsigned                stream_id:16;
        unsigned                no_interrupt:1;
        unsigned                zero:1;
        unsigned                short_not_ok:1;
        unsigned                dma_mapped:1;
        void                    (*complete)(struct usb_ep *ep, struct usb_request *req);
        void                    *context;
        struct list_head        list;
        unsigned                frame_number;
        int                     status;
        unsigned                actual;
};
```

buf	: Transfer buffer address
length	: Data length of transmission and reception
dma	: DMA address structure pointer
sg	: A scatter list for SG(Scatter/Gather)-capable controllers
num_sgs	: Number of SG entries
num_mapped_sgs	: Number of SG entries mapped to DMA (internal)
stream_id	: The stream id, when USB3.0 bulk streams are being used
no_interrupt	: Transmission discontinuation unnecessary flag
zero:1	: Zero length packet addition flag
short_not_ok:1	: Short packet improper flag
dma_mapped:1	: Indicates if request has been mapped to DMA (internal)
complete()	: Completion callback function pointer
context	: transfer context
list	: list structure (Write-protected)
frame_number	: The interval number in (micro) frame in which the isochronous transfer was transmitted or received.
status	: status of transfer (Write-protected)
actual	: The completion data length of transmission (Write-protected)

4.2.2 usb_ep_ops structure

```
struct usb_ep_ops {
        int                     (*enable) (struct usb_ep *ep, const struct usb_endpoint_descriptor *desc);
        int                     (*disable) (struct usb_ep *ep);
        void                    (*dispose) (struct usb_ep *ep);
        struct usb_request      *(*alloc_request) (struct usb_ep *ep, gfp_t gfp_flags);
        void                    (*free_request) (struct usb_ep *ep, struct usb_request *req);
        int                     (*queue) (struct usb_ep *ep, struct usb_request *req, gfp_t gfp_flags);
        int                     (*dequeue) (struct usb_ep *ep, struct usb_request *req);
        int                     (*set_halt) (struct usb_ep *ep, int value);
        int                     (*set_wedge) (struct usb_ep *ep);
        int                     (*fifo_status) (struct usb_ep *ep);
        void                    (*fifo_flush) (struct usb_ep *ep);
};
```

enable()	: Endpoint enable function pointer
disable()	: Endpoint disable function pointer
dispose()	: Endpoint dispose function pointer
alloc_request()	: usb_request structure acquisition pointer
free_request()	: usb_request structure release function pointer
queue()	: Transfer request function pointer
dequeue()	: Transfer stop function pointer
set_halt()	: Setting status of STALL function pointer
set_wedge()	: Setting status of STALL function pointer
fifo_status()	: Status of FIFO acquisition function pointer
fifo_flush()	: FIFO FLUSH function pointer

4.2.3 usb_ep_caps structure

```
struct usb_ep_caps {
          unsigned                  type_control:1;
          unsigned                  type_iso:1;
          unsigned                  type_bulk:1;
          unsigned                  type_int:1;
          unsigned                  dir_in:1;
          unsigned                  dir_out:1;
};
```

type_control	: Endpoint supports control type (reserved for ep0)
type_iso	: Endpoint supports isochronous transfers
type_bulk	: Endpoint supports bulk transfers
type_int	: Endpoint supports interrupt transfers
dir_in	: Endpoint supports IN direction
dir_out	: Endpoint supports OUT direction

4.2.4 usb_ep structure

```
struct usb_ep {
          void                          *driver_data;
          const char                    *name;
          const struct usb_ep_ops       *ops;
          struct list_head              ep_list;
          struct usb_ep_caps            caps
          bool                          claimed
          bool                          enabled
          unsigned                      maxpacket:16;
          unsigned                      maxpacket_limit:16
          unsigned                      max_streams:16;
          unsigned                      mult:2;
          unsigned                      maxburst:5;
          u8                            address;
          const struct                  usb_endpoint_descriptor *desc;
          const struct                  usb_ss_ep_comp_descriptor *comp_desc;

};
```

driver_data : driver data

name : Endpoint name

ops : USB Endpoint operation function structure

ep_list : list structure(Write-protected)

caps : The structure describing types and directions supported by endpoint.

claimed : This is used autoconfig.

enabled : This is used usb_enable/disable_endpoint.

maxpacket:16 : MaxPacketSize(Write-protected)

maxpacket_limit:16 : MaxPacketSize(Write-protected)

max_stream:16 : The maximum number of streams supported by this EP (0 - 16, actual number is 2^n)

mult:2 : multiplier, 'mult' value for SS Isoc EPs

maxburst:5 : the maximum number of bursts supported by this EP (for usb3)

address : used to identify the endpoint when finding descriptor that matches connection speed

desc : endpoint descriptor. This pointer is set before the endpoint is enabled and remains valid until the endpoint is disabled.

comp_desc : In case of Super Speed support, this is the endpoint companion descriptor that is used to configure the endpoint

4.2.5 usb_gadget_ops structure

```
struct usb_gadget_ops {
        int                 (*get_frame)(struct usb_gadget *);
        int                 (*wakeup)(struct usb_gadget *);
        int                 (*set_selfpowered) (struct usb_gadget *, int is_selfpowered);
        int                 (*vbus_session) (struct usb_gadget *, int is_active);
        int                 (*vbus_draw) (struct usb_gadget *, unsigned mA);
        int                 (*pullup) (struct usb_gadget *, int is_on);
        int                 (*ioctl)(struct usb_gadget *,unsigned code, unsigned long param);
        void                (*get_config_params)(struct usb_dcd_config_params *);
        int                 (*udc_start)(struct usb_gadget *, struct usb_gadget_driver *);
        int                 (*udc_stop)(struct usb_gadget *, struct usb_gadget_driver *);

        void                (*udc_set_speed)(struct usb_gadget *, enum usb_device_speed);

        struct usb_ep       (*match_ep)(struct usb_gadget *,
                            struct usb_endpoint_descriptor *,
                            struct usb_ss_ep_comp_descriptor * );
};
```

get_frame()	: Frame number acquisition function pointer
wakeup()	: Remote wake up function pointer
set_selfpowered()	: Self / Bus power setting function pointer
vbus_session()	: VBUS Status notification function pointer
vbus_draw()	: Consumption current notification function pointer
pullup()	: Pull-up status control function pointer
ioctl()	: IO Control function pointer
get_config_params()	: get_config_params function pointer
udc_start()	: UDC start function pointer
udc_stop()	: UDC stop function pointer
udc_set_speed()	: UDC set speed function pointer
match_ep()	: UDC match ep function pointer

4.2.6 usb_gadget structure

```
struct usb_gadget {
            struct work_struct             work;
            struct usb_udc                 *udc;
            /* readonly to gadget driver */
            const struct usb_gadget_ops    *ops;
            struct usb_ep                  *ep0;
            struct list_head               ep_list;    /* of usb_ep */
            enum usb_device_speed          speed;
            enum usb_device_speed          max_speed;
            enum usb_device_state          state;
            const char                     *name;
            struct device                  dev;
            unsigned                       isoch_delay;
            unsigned                       out_epnum;
            unsigend                       in_epnum;
            unsigned                       mA;
            struct usb_otg_caps            *otg_caps
            unsigned                       sg_supported:1;
            unsigned                       is_otg:1;
            unsigned                       is_a_peripheral:1;
            unsigned                       b_hnp_enable:1;
            unsigned                       a_hnp_support:1;
            unsigned                       a_alt_hnp_support:1;
            unsigned                       hnp_polling_support:1;
            unsigned                       host_request_flag:1;
            unsigned                       quirk_ep_out_aligned_size:1;
            unsigned                       quirk_altset_not_supp:1;
            unsigned                       quirk_stall_not_supp:1;
            unsigned                       quirk_zlp_not_supp:1;
            unsigned                       is_selfpowered:1;
            unsigned                       deactivated:1;
            unsigned                       connected:1;
            unsigned                       lpm_capable:1;
            unsigned                       irq;
};
```

ops : The operation structure for Gadget

ep0 : Endpoint_0 information structure

ep_list : usb_ep storing list

speed : USB operation speed

max_speed : Maximal speed the UDC can handle. UDC must support this and all slower speeds.

state : the state we are now (attached, suspended, configured, etc)

name : device name

dev : device structure

isoch_delay : value from Set Isoch Delay request. Only valid on SS/SSP

out_epnum	: last used out ep number
in_epnum	: last used in ep number
mA	: last set mA value
*otg_caps	:OTG capabilities of this gadget
sg_supported	: true if we can handle scatter-gather
is_otg:1	: OTG supported flag
is_a_peripheral:1	: A_PERIPHERAL status flag when OTG operate
b_hnp_enable:1	: B_HNP_ENABLE status flag when OTG operate
a_hnp_support:1	: A_HNP_SUPPORT status flag when OTG operate
hnp_polling_support:1	: HNP_POLLING status flag when OTG operate
host_request_flag:1	: indicating if A-Peripheral or B-Peripheral wants to take host role
a_alt_hnp_support:1	: A_ALT_HNP_SUPPORT status flag when OTG operate
quirk_ep_out_aligned_size:1	: ep out requires buffer size to be aligned to MaxPacketSize
quirk_altset_not_supp:1	: alt set not supported flag
quirk_stall_not_supp:1	: stall not supported flag
quirk_zlp_not_supp:1	: zero length packet not supported flag
is_selfpowered:1	: gadget is self-powered
deactivated:1	: True if gadget is deactivated - in deactivated state it cannot be connected
connected:1	: True if gadget is connected
lpm_capable:1	: LPM support status flag
irq	: the interrupt number for device controller

4.2.7 usb_gadget_driver structure

```
struct usb_gadget_driver {
        char                        *function;
        enum usb_device_speed       max_speed;
        int                         (*bind)(struct usb_gadget *gadget, struct usb_gadget_driver *driver);
        void                        (*unbind)(struct usb_gadget *);
        int                         (*setup)(struct usb_gadget *, const struct usb_ctrlrequest *);
        void                        (*disconnect)(struct usb_gadget *);
        void                        (*suspend)(struct usb_gadget *);
        void                        (*resume)(struct usb_gadget *);
        void                        (*reset)(struct usb_gadget *);

        struct device_driver        driver;
        char                        *udc_name;
        struct list_head            pending;
        unsigned                    match_existing_only:1;
};
```

function : function name

max_speed : Max operation speed

bind() : Callback function pointer at the time of registration

unbind() : Callback function pointer at the time of registration release

setup() : Callback function pointer at the time of setup reception

disconnect() : Callback function pointer at the time of USB disconnecting

suspend() : Callback function pointer at the time of USB suspended

resume () : Callback function pointer at the time of USB resumed

driver : device driver structure

reset () : Callback function pointer at the time of USB reset

driver : device driver structure

udc_name : device driver name

pending : It is used for deferred probe.

match_existing_only : If udc is not found, return an error and don't add this gadget driver to list of pending

4.3 Global Variables and Constants

4.3.1 Global variables

There are no global variables for this module.

4.3.2 Global constants

There are no global constants for this module.

5. Integration

5.1 Directory Configuration

The directory configuration is shown below.

```
    —— drivers/usb/gadget/udc/renesas_usb3.c
```

Figure 5-1 Directory Configuration

5.1.1 Integration of a USB Function control driver for R-Car (H3/M3/M3N/E3)

To enable the function of this module, make the following setting with Kernel Configuration. When using only USB3.0 function, setting of USB3.0 host is unnecessary.

```
Device Drivers    --->
      [*]   USB support --->
              ...
              <    > xHCI HCD (USB 3.0) support
              <    > xHCI support for Renesas R-Car SoCs

              <*>    USB Gadget Support --->
                     <*>    USB Peripheral Controller --->
                            <*>    Renesas USB3.0 Peripheral controller

            PHY Subsystem --->
                     <*>    Renesas R-Car generation 3 USB 3.0 PHY driver
```

Figure 5-2 Kernel configuration for this module (H3/M3/M3N/E3)

5.1.2 Integration of a USB Function control driver for Role Swap

To enable the role swap of this module, make the following setting with Kernel Configuration.
Refer to “Linux Interface Specification Device Driver USB 3.0 Host User’s Manual: Software” for firmware acquisition and configuration.

```
Device Drivers    --->
           Generic Driver Options --->
              (r8a779x_usb3_v3.dlmem) External firmware blobs to build into kernel binary
              (firmware) Firmware blobs root directory

      [*]   USB support --->
              <*> xHCI HCD (USB 3.0) support
              <*> xHCI support for Renesas R-Car SoCs

              ...
              <*>    USB Gadget Support --->
                     <*>    USB Peripheral Controller --->
                            <*>    Renesas USB3.0 Peripheral controller

            PHY Subsystem --->
                     <*>    Renesas R-Car generation 3 USB 3.0 PHY driver
```

Figure 5-3 Kernel configuration for Role Swap

5.1.3 Integration of a USB gadget driver

The example of integration of a USB gadget driver is shown below.

Please perform the following setup, when you integrate Mass Storage Gadget as module.

1. Please enable (input "M") the following item in "USB Gadget Support".

```
Device Drivers    --->
        [*]    USB support --->
                <*>    USB Gadget Support --->
                        <M>      USB Gadget precomposed configurations
                        <M>         Mass Storage Gadget
```

Figure 5-4 Kernel configuration for USB Mass Storage Gadget driver

5.2 Option Setting

5.2.1 Module Parameters

There are no module parameters.

5.2.2 Kernel Parameters

There are no kernel parameters.

6. USB 3.0 Role Swap Interface

This module supports USB 3.0 Role Swap Interface. This module supports only role swap don't using Host Negotiation Protocol (HNP). No support for Session Request Protocol (SRP).

6.1 Role Swap usage

This section shows an example using two System evaluation boards board (A) and board (B), and at first board (A) acts as a Host, and then switch to the Function as follows.

1) Board (A) (B), both to enable functionality of the USB 3.0 Host and USB 2.0 Function (for example, USB Mass Storage Gadget).

An attention is needed to use USB gadget driver. If USB 2.0 Function is enabled, you need to first load the USB gadget driver into the USB 2.0 port. An example of setting USB Mass Storage Gadget driver to USB3.0 Function is as follows.

Board (A):

```
insmod g_xxx.ko --- (USB2.0 ch0 CN9 is associated 1st gadget)

insmod g_yyy.ko --- (USB2.0 ch3 CN37 is associated 2nd gadget) (R-Car H3 only)

insmod g_mass_storage.ko --- (USB3.0 ch0 CN11 is associated 3rd gadget)
```

Board (B):

```
insmod g_xxx.ko --- (USB2.0 ch0 CN9 is associated 1st gadget)
```

2) Connect as follows.

Board(A) – A plug (CN11)– usb cable(A- Micro B cable) – Micro B plug (CN9) – Board(B)

3) In Board (A) and run the following command.

```
echo peripheral > /sys/devices/platform/soc/ee020000.usb/role
```

4) In Board (B) and run the following command.

```
echo host > /sys/devices/platform/soc/ee080200.usb-phy/role
```

5) Board (A) is Function, Board (B) will be switched to the Host.
Please note that Board(A) must input the following command if you want the board to act as the Host again.

```
echo host > /sys/devices/platform/soc/ee020000.usb/role
```

6.2 About Super Speed Connection after Switching from USB Function Mode to USB Host Mode

After the procedure of 6.1, when you disconnect the cable and change to Host and connect the cable with a Super Speed compatible peripheral device, you will need to initialize USB 3.0 Host because it will connect with High-speed. Execute the following command, if you want to connect with Super Speed.

```
1) echo 'ee000000.usb' > /sys/bus/platform/drivers/xhci-hcd/unbind
2) echo 'ee000000.usb' > /sys/bus/platform/drivers/xhci-hcd/bind
```

7. How to use USB3.0 Function

R-Car H3/M3/M3N/E3 supports Super Speed of USB3.0 Function. But the USB3.0 CN11 connector of Salvator-X and Salvator-XS and Ebisu board is for USB Host. This connector outputs 5V of VBUS, since the ID pin of R-Car is not connected. By using b_device, the mode changes USB Function mode and 5V of VBUS output is disconnected. If you want to use this connector as USB3.0 Function, please follow the steps below.

7.1 USB Function mode usage

1) Refer to 5.1.1 Kernel configuration for build.

2) insmod loads gadget.

An attention is needed to use USB gadget driver. If USB 2.0 Function is enabled, you need to first load the USB gadget driver into the USB 2.0 port. An example of setting USB Mass Storage Gadget driver to USB3.0 Function is as follows.

```
insmod g_xxx.ko --- (USB2.0 ch0 CN9 is associated 1st gadget)

insmod g_yyy.ko --- (USB2.0 ch3 CN37 is associated 1st gadget) (R-Car H3 only)

insmod g_ mass_storage.ko --- (USB3.0 ch0 CN11 is associated 2nd gadget)
```

3) Enable Super Speed (b-device)

```
echo 1 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

4) Connect the USB cable

5) When using the E3 board, please run the following command continues.

```
echo 2 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

7.2 How to change to the USB Function mode from USB 3.0 Host Driver

This module does not support roll swap with Super Speed. The system evaluation board initially acts as a Host. The way to switch to the function is as follows.

1) Refer to 5.1.2 Kernel configuration for build.

2) Disable xHCI driver.

[using as USB3.0 Func port]

```
  echo 'ee000000.usb' > /sys/bus/platform/drivers/xhci-hcd/unbind
```

3) Enable functionality of the USB 3.0 Function (for example, USB Mass Storage Gadget).

An attention is needed to use USB gadget driver. If USB 2.0 Function is enabled, you need to first load the USB gadget driver into the USB 2.0 port. An example of setting USB Mass Storage Gadget driver to USB3.0 Function is as follows.

```
  insmod g_xxx.ko --- (USB2.0 ch0 CN9 is associated 1st gadget)

  insmod g_yyy.ko --- (USB2.0 ch3 CN37 is associated 2nd gadget)(R-Car H3 only)

  insmod g_mass_storage.ko--- (USB3.0 ch0 CN11 is associated 3rd gadget)
```

4) Run the following command.

[using as USB3.0 Func port]

```
echo 1 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

After executing the command, connect the USB cable.

5) When using the E3 board, please run the following command continues.

[using as USB3.0 Func port]

```
echo 2 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

6) If you want the USB 3.0 connector (CN11) to act as a host again, enabel xHCI driver by entering the following command after disconnection of usb cable.

[using as USB3.0 Host port]

```
echo 0 > /sys/kernel/debug/usb/ee020000.usb/b_device
echo 'ee000000.usb' > /sys/bus/platform/drivers/xhci-hcd/bind
```

7.3 How to disconnect and re-connect the USB cable for R-Car E3

There is no VBUS detection pin in R-Car E3.

So, disconnect and connect the usb cable according to the following procedure.

1) Disconnect usb cable.

2) Execute below command.

```
echo 1 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

3) Connect usb cable.

4) Execute below command.

```
echo 2 > /sys/kernel/debug/usb/ee020000.usb/b_device
```

REVISION HISTORY		Linux Interface Specification Device Driver USB 3.0 Function User's Manual: Software	
Rev.	**Date**	**Description**	
		Page	**Summary**
0.1	Jun. 14, 2017	—	New creation.
1.00	Aug. 8, 2017	—	Update document format.
		2,3	1.6 Notice Update support device. Add notes of role swap setting.
1.01	Oct. 24, 2017	All	Add R-Car M3N support.
		1	1.2 Function Change the maximum PIPE is 30 to 5 except PIPE0 and EP0.
		2	Update related documents.
1.50	Jan. 29, 2018	2	Update related documents.
		30	Update "Integration of a USB gadget driver".
		31	6.1 Role Swap usage Update usage of USB gadget driver.
1.51	Mar. 28, 2018	All	Add R-Car E3 support.
1.52	Oct. 29, 2018	2	1.4.2 Related Documents Update Related Documents
		3	1.6 Notice Add Target device.
		28	4.2.6 usb gadget structure Fix the quirk_zlp_not_supp:1 and is_selfpowered:1 of explanation.
		30	5.1.1 Integration of a USB Function control driver for R-Car (H3/M3/M3N) Add Kernel configuration figure 5.1.2 Integration of a USB Function control driver for R-Car E3 Add Kernel configuration figure
		33	7.USB3.0 Function Super-Speed Interface Add notes of Super-Speed Usage.
2.00	Dec. 25, 2018	2	1.4.2 Related Documents Update Related Documents
		3	1.6 Notice Add notes of USB3.0 Function support.
		5	3.1 Hardware Environment Update Hardware Environment Add Ebisu-4D Board
		30	5.1.1 Integration of a USB Function control driver for R-Car (H3/M3/M3N/E3) Add Kernel configuration figure
		31	6.1 Role Swap usage Update Role Swap usage
		33	7.1 Super-Speed usage Update Super-Speed usage 7.2 How to disconnect and re-connect the USB cable for R-Car E3 New creation.
		-	Update AddressList

REVISION HISTORY		Linux Interface Specification Device Driver USB 3.0 Function User’s Manual: Software	
Rev.	Date	Description	
		Page	Summary
2.01	Apr. 17, 2019	2	1.4.2 Related Documents Update Table 1.3 Related Documents
		2	1.6 Notice Update Notice
		29	5.1.1 Integration of a USB Function control driver for R-Car (H3/M3/M3N/E3) Update configuration option
		29	5.1.2 Integration of a USB Function control driver for Role Swap Add a reference to the user manual of the USB 3.0 host driver.
		30	6.USB 3.0 Role Swap Interface In this chapter, we returned to the high-speed procedure (Ver 1.52).
		30	6.2 About Super Speed Connection after Switching from USB Function Mode to USB Host Mode Add procedure after Roll Swap.
		31	7.1 USB Function mode usage Change chapter title to "USB Function mode usage” from “Super-Speed usage”
		31	7.2 How to change to the USB Function mode from USB 3.0 Host Driver Add using USB 3.0 Function from USB 3.0 Host Driver Enabled State.
		All	Standardize on "Super Speed"
		-	Update AddressList
2.50	Apr. 21, 2021	2	1.4.2 Related Documents Update Revision of Hardware User’s Manual
		21, 22, 24, 25, 26	Update USB structures
		31, 32	Update USB debugfs directory
3.00	Dec. 10, 2021	-	Add Kernel v5.10.41 support

Linux Interface Specification Device Driver USB 3.0 Function
User’s Manual: Software

Publication Date:	Rev.0.1	Jun. 14, 2017
	Rev.3.00	Dec. 10, 2021
Published by:	Renesas Electronics Corporation	

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc. Milpitas Campus
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics America Inc. San Jose Campus
6024 Silver Creek Valley Road, San Jose, CA 95138, USA
Tel: +1-408-284-8200, Fax: +1-408-284-2775

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

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 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, 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 200333, China
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, #06-02 Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0

RENESAS

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

Colophon 3.1

Linux Interface Specification
Device Driver
USB 3.0 Function

Renesas Electronics Corporation