# Overview

## Overview

This manual explains the DMA Engine device driver in R-Car H3/M3/M3N/E3/D3/V3U/V3H Linux.

## Support DMAC

Table 1-1 Support DMAC (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| DMAC | Overview | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| R-Car H3 | R-Car M3/M3N | R-Car E3 | R-Car D3 | R-Car V3U | R-Car V3H |
| SYS-DMAC | 48 channels | | 48 channels | 24 channels | 24 channels | 32 channels |
| Audio-DMAC | 32 channels | | 16 channels | 16 channels | - | - |
| USBHS-DMAC | 8 channels | 4 channels | 4 channels | 4 channels | - | - |
| RT-DMAC | 16 channels | | 16 channels | 8 channels | 64 channels | 32 channels |

## Function

This module controls DMAC on R-Car H3/M3/M3N/E3/D3/V3U/V3H with using DMA Engine framework, and provides the following functions.

* DMA device driver module management with DMA Engine framework.
* Control each channel of support DMAC. (Refer to Table 1-1 and Table 1-2 in detail)
* Call the registered callback function when DMA transfer is completed.
* Control the data transfer between the memory and the peripheral. (SYS-DMAC, Audio-DMAC, USBHS-DMAC, and RT-DMAC)
* Forced to terminate during transmission.
* Control the descriptor transfer

**Table 1-2 Support status of SYS-DMAC**

| **Feature** | **SW support status** | |
| --- | --- | --- |
| **R-Car H3/M3/M3N/E3/D3/V3H** | **R-Car V3U** |
| R-Car H3/M3/M3N/E3: Up to 48 channels  R-Car D3: Up to 24 channels  R-Car V3H: Up to 32 channels  R-Car V3U: Up to 24 channels | Supported | Supported |
| Transfer data length:  Byte, word (2 bytes), longword (4 bytes), 8 bytes, 32 bytes, and 64 bytes | Supported | Supported |
| Transfer request: Requests from on-chip peripheral modules or auto requests can be selected | Supported | Supported |
| Bus mode: Selectable from normal mode and slow mode | Not supported  (Fixed normal mode) | Supported (\*) |
| Priority: Either fixed priority or round-robin arbitration can be selected for use in arbitration among the transfer channels | Not supported  (Fixed ‘fixed priority’) | Not supported  (Fixed ‘fixed priority’) |

Note: (\*): In the slow speed mode, RRATE\_RD and RRATE\_WR registers are used to control.

**Table 1-3 Support status of USB-DMAC**

| **Feature** | **SW support status** | |
| --- | --- | --- |
| **R-Car H3/M3/M3N/E3/D3** | **R-Car V3U/V3H** |
| Supports two channels that can work concurrently | Supported | Not available |
| Interrupts:  - USB-DMAC is composed of USB-DMAC0/2 and USB-DMAC1/3. There are 2 channels in USB-DMAC0/2 and 2 channels in USB-DMAC1/3. (USB-DMAC2 and USB-DMAC3 are only R-Car H3.)  - Transfer end interrupt upon receiving a short packet from the HS-USB  - Interrupts can also be generated when a NULL packet is received or an address error is detected | Supported | Not available |
| Timeout interrupt: A timeout interrupt can be generated after the specified cycles after the last HS-USB transfer request. (This interrupt is used for the timeout when the final packet received is Max packet.) | Not supported | Not available |

**Table 1-4 Support status of Audio-DMAC**

| **Feature** | **SW support status** | |
| --- | --- | --- |
| **R-Car H3/M3/M3N/E3/D3** | **R-Car V3U/V3H** |
| R-Car H3/M3/M3N: Up to 32 channels  R-Car E3/D3: Up to 16 channels | Supported | Not available |
| Transfer data length: Byte, word (2 bytes), longword (4 bytes), 8 bytes, 32 bytes, and 64 bytes | Supported | Not available |
| Transfer request: Requests from on-chip peripheral modules or auto requests can be selected | Supported | Not available |
| Bus mode: Selectable from normal mode and slow mode | Not supported (Fixed normal mode) | Not available |
| Priority: Either fixed priority or round-robin arbitration can be selected for use in arbitration among the transfer channels | Not supported  (Fixed ‘fixed priority’) | Not available |

**Table 1-5 Support status of RT-DMAC**

| **Feature** | **SW support status** | |
| --- | --- | --- |
| **R-Car H3/M3/M3N/E3/D3/V3H** | **R-Car V3U** |
| R-Car H3/M3/M3N/E3: Up to 16 channels  R-Car D3: Up to 8 channels  R-Car V3H: Up to 32 channels  R-Car V3U: Up to 64 channels | Supported | Supported |
| Transfer data length: Byte, word (2 bytes), longword (4 bytes), 8 bytes, 32 bytes, and 64 bytes | Supported | Supported |
| Transfer request: Requests from on-chip peripheral modules or auto requests can be selected | Supported | Supported |
| Bus mode: Selectable from normal mode and slow mode | Not supported  (Fixed normal mode) | Supported (\*) |
| Priority: Either fixed priority or round-robin arbitration can be selected for use in arbitration among the transfer channels | Not supported  (Fixed ‘fixed priority’) | Not supported  (Fixed ‘fixed priority’) |

Note: (\*): In the slow speed mode, RRATE\_RD and RRATE\_WR registers are used to control.

## Reference

### Standards

There is no reference document on standards.

### Related Documents

The following table shows the document related to this module.

**Table 1-6　　Related documents (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

| **Number** | **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 System Evaluation Board Salvator-X Hardware Manual RTP0RC7795SIPB0011S | Rev.1.03 | Jul. 19, 2016 |
| - | Renesas Electronics | R-CarM3-SiP System Evaluation Board Salvator-X Hardware Manual RTP0RC7796SIPB0011S | Rev.0.03 | Jul. 19, 2016 |
| - | 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-CarE3 System Evaluation Board Ebisu  Hardware Manual RTP0RC77990SEB0010S | Rev.0.01 | Mar. 9, 2018 |
| - | Renesas Electronics | R-CarE3 System Evaluation Board Ebisu-4D (E3 board 4xDRAM) Hardware Manual | Rev.1.01 | Jul. 19, 2018 |
| - | Renesas Electronics | R-CarD3 System Evaluation Board Hardware Manual RTP0RC77995SEB0010S | Rev.1.20 | Jul. 25, 2017 |
| - | Renesas Electronics | R-Car V3U series User’s Manual | Rev.0.5 | Jul. 31, 2020 |
| - | Renesas Electronics | R-CarV3U System Evaluation Board Falcon Hardware Manual | Rev.0.01 | Sep. 11, 2020 |
| - | Renesas Electronics | R-Car V3H\_2 Additional Document for User’s Manual: Hardware | Rev.0.50 | Jul. 31, 2020 |
| - | Renesas Electronics | R-CarV3H System Evaluation Board Condor-I Hardware Manual | Rev.0.02 | Nov. 11, 2019 |

## Restrictions

There is no restriction in this module.

# Terminology

The following table shows the terminology related to this module.

**Table 2-1 Terminology**

| **Terms** | **Explanation** |
| --- | --- |
| DMA | Direct Memory Access |
| DMAC | DMA Controller |
| RapidIO | Standard for inter-system connection |

# Operating Environment

## Hardware Environment

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

**Table 3-1 Hardware specification (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

| **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 |
| R-CarV3U System Evaluation Board Falcon | - | Renesas Electronics |
| R-CarD3 System Evaluation Board Draak | - | Renesas Electronics |
| R-CarV3H System Evaluation Board Condor-I | - | Renesas Electronics |

## Module Configuration

The following figures show the configuration of this module.

Kernel Module

**This module**

DMA Engine driver

**Kernel mode**

**Hardware**

DMA Engine Interface

Audio Module

I2C/SCIF/MSIOF/

TPU/TSIF/DRIF/

CANFD

DMA Controller

(SYS-DMAC)

USB-IP

CANFD/ADC/

MSIOF/I2C

DMA Controller

(Audio-DMAC) (\*)

DMA Controller

(USB-DMAC) (\*)

DMA Controller

(RT-DMAC)



Figure 3-1 　DMA Engine Driver configuration (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

## State Transition Diagram

The state transition managed by this module has "DMA stop" state and "During DMA transmission" state. This module does not support the suspend state.

device\_issue\_pending()

Auto or device\_terminate\_all()

Figure 3-2 DMA Engine Driver State Transition Diagram (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

# External Interface

This module provides the interface for the kernel space that is used DMA Engine framework. There are no interfaces for the user space in this module.

## Device Node

There are no device nodes in this module.

## External Function

This section explains in the following format about the functions this module supplies.

|  |  |
| --- | --- |
| [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. |
|  |  |

Table 4-1 External function (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| **Chapter** | **Function name** | **Description** |
| --- | --- | --- |
| 4.2.1 | dma\_request\_channel | Allocate and get DMA channel. |
| 4.2.2 | dma\_request\_chan | Allocate an exclusive slave channel |
| 4.2.3 | dma\_request\_chan\_by\_mask | Allocate a channel satisfying certain capabilities |
| 4.2.4 | dma\_request\_slave\_channel | Allocate an exclusive slave channel |
| 4.2.5 | dma\_async\_device\_register | Register the device correspond to DMA function to the system. |
| 4.2.6 | dma\_async\_device\_unregister | Delete the device correspond to DMA function from the system. |
| 4.2.7 | dma\_async\_tx\_descriptor\_init | Initialize the descriptor. |
| 4.2.8 | dma\_find\_channel | Search the channel of the specified transaction type. |
| 4.2.9 | dma\_issue\_pending\_all | Execute the request if execution of the pending request is possible. |
| 4.2.10 | dma\_release\_channel | Release the channel. |
| 4.2.11 | dma\_run\_dependencies | Perform the dependence processing before execution of a target channel. |
| 4.2.12 | dma\_submit\_error | Check the DMA request cookie. |
| 4.2.13 | dma\_sync\_wait | Perform waiting process for the completion of transmission. |
| 4.2.14 | dma\_wait\_for\_async\_tx | Perform waiting process for the completion of transmission (with a descriptor check). |
| 4.2.15 | dmaengine\_get | Enable to use a free channel with the DMA Engine interface. |
| 4.2.16 | dmaengine\_put | Release the channel that was enabled by the call of dmaengine\_get(). |
| 4.2.17 | dma\_cap\_clear | Clear the DMA transaction type. |
| 4.2.18 | dma\_cap\_set | Set the DMA transaction type. |
| 4.2.19 | dma\_cap\_zero | Initialize the DMA transaction type. |
| 4.2.20 | dma\_has\_cap | Check whether holds the DMA transaction type. |

### dma\_request\_channel

|  |  |  |
| --- | --- | --- |
| [Overview] | Allocate and get DMA channel | |
|  |  | |
| [Function Name] | dma\_request\_channel | |
|  |  | |
| [Calling format] | struct dma\_chan \*dma\_request\_channel(dma\_cap\_mask\_t mask, dma\_filter\_fn fn, | |
|  | void \*fn\_param); | |
|  |  | |
| [Arguments] | mask | capabilities that the channel must satisfy |
|  | fn | optional callback to disposition available channels |
|  | fn\_param | opaque parameter to pass to dma\_filter\_fn |
|  |  | |
| [Returns] | not NULL | Success (channel structure address) |
|  | NULL | Error |
|  |  | |
| [Feature] | Get the exclusive channel which satisfies the conditions specified by the 1st argument mask. | |
|  | The definition of the callback function of the 2nd argument is the following format. | |
|  | This function is not the transmission completion callback but the filter option callback. | |
|  | bool (\*dma\_filter\_fn)(struct dma\_chan \*chan, void \*filter\_param); | |
|  | Support transaction type is follow.   |  |  | | --- | --- | | **Definition Name** | **Support** | | DMA\_MEMCPY | Support | | DMA\_XOR | Not support | | DMA\_PQ | Not support | | DMA\_XOR\_VAL | Not support | | DMA\_PQ\_VAL | Not support | | DMA\_MEMSET | Not support | | DMA\_MEMSET\_SG | Not support | | DMA\_INTERRUPT | Not support | | DMA\_PRIVATE | Support | | DMA\_ASYNC\_TX | Support | | DMA\_SLAVE | Support | | DMA\_CYCLIC | Not support | | DMA\_INTERLEAVE | Not support | | DMA\_TX\_TYPE\_END | Not support | | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_request\_chan

|  |  |  |
| --- | --- | --- |
| [Overview] | Allocate an exclusive slave channel | |
|  |  | |
| [Function Name] | dma\_request\_chan | |
|  |  | |
| [Calling format] | struct dma\_chan \*dma\_request\_chan(struct device \*dev, const char \*name) | |
|  |  | |
| [Arguments] | dev | pointer to client device structure |
|  | name | slave channel name |
|  |  |  |
|  |  | |
| [Returns] | Channel structure address | Success |
|  | -EPROBE\_DEFER | Error |
|  |  | |
| [Feature] | Get the exclusive channel which satisfies the conditions specified by device tree. | |
|  | Device tree property detail are please see 4.6.1. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_request\_chan\_by\_mask

|  |  |  |
| --- | --- | --- |
| [Overview] | Allocate a channel satisfying certain capabilities | |
|  |  | |
| [Function Name] | dma\_request\_chan\_by\_mask | |
|  |  | |
| [Calling format] | struct dma\_chan \*dma\_request\_chan\_by\_mask(const dma\_cap\_mask\_t \*mask) | |
|  |  | |
| [Arguments] | mask | capabilities that the channel must satisfy. |
|  |  |  |
|  |  |  |
|  |  | |
| [Returns] | NULL | Error |
|  | -ENODEV | No such device (Argument mask is NULL) |
|  | Other value | Success (channel structure address) |
|  |  |  |
|  |  | |
| [Feature] | Get the exclusive channel which satisfies the conditions specified by argument mask. | |
|  | Set the DMA transaction type (target bit) to the argument mask with the dma\_cap\_set | |
|  | function and use it as the argument mask of this function. | |
|  | Support transaction type is shown below.   |  |  | | --- | --- | | **Definition Name** | **Support** | | DMA\_MEMCPY | Support | | DMA\_XOR | Not support | | DMA\_PQ | Not support | | DMA\_XOR\_VAL | Not support | | DMA\_PQ\_VAL | Not support | | DMA\_MEMSET | Not support | | DMA\_MEMSET\_SG | Not support | | DMA\_INTERRUPT | Not support | | DMA\_PRIVATE | Support | | DMA\_ASYNC\_TX | Support | | DMA\_SLAVE | Support | | DMA\_CYCLIC | Not support | | DMA\_INTERLEAVE | Not support | | DMA\_TX\_TYPE\_END | Not support | | |
|  |  | |
| [Remark] | Refer to 4.5.2.2 about dma\_transaction\_type. | |
|  |  | |
|  |  | |

### dma\_request\_slave\_channel

|  |  |  |
| --- | --- | --- |
| [Overview] | Allocate an exclusive slave channel | |
|  |  | |
| [Function Name] | dma\_request\_slave\_channel | |
|  |  | |
| [Calling format] | struct dma\_chan \*dma\_request\_slave\_channel(struct device \*dev, const char \*name); | |
|  |  | |
| [Arguments] | dev | pointer to client device structure |
|  | name | slave channel name |
|  |  |  |
|  |  | |
| [Returns] | not NULL | Success (channel structure address) |
|  | NULL | Error |
|  |  | |
| [Feature] | This function wrapped dma\_request\_chan() function. | |
|  | In this function check return value of dma\_request\_chan() function, and it returns address of channel structure or NULL. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_async\_device\_register

|  |  |  |
| --- | --- | --- |
| [Overview] | Register the device correspond to DMA function to the system | |
|  |  | |
| [Function Name] | dma\_async\_device\_register | |
|  |  | |
| [Calling format] | int dma\_async\_device\_register(struct dma\_device \*device); | |
|  |  | |
| [Arguments] | device | File descriptor |
|  |  | |
| [Returns] | 0 | Success |
|  | -EIO | I/O error |
|  | -ENODEV | No such device (Argument device is NULL) |
|  | -ENOMEM | Out of memory |
|  | -ENOSPC | No space left on device |
|  |  | |
| [Feature] | Register the device correspond to DMA function to the system. | |
|  | This function is called from the DMA driver initialization process. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_async\_device\_unregister

|  |  |  |
| --- | --- | --- |
| [Overview] | Delete the device correspond to DMA function from the system | |
|  |  | |
| [Function Name] | dma\_async\_device\_unregister | |
|  |  | |
| [Calling format] | void dma\_async\_device\_unregister(struct dma\_device \*device); | |
|  |  | |
| [Arguments] | device | device information to delete |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Delete the device correspond to DMA function from the system. | |
|  | This function is called from the DMA driver termination process. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_async\_tx\_descriptor\_init

|  |  |  |
| --- | --- | --- |
| [Overview] | Initialize the descriptor | |
|  |  | |
| [Function Name] | dma\_async\_tx\_descriptor\_init | |
|  |  | |
| [Calling format] | void dma\_async\_tx\_descriptor\_init(struct dma\_async\_tx\_descriptor \*tx, | |
|  | struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | tx | descriptor to register the channel information |
|  | chan | channel information to register to the descriptor |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Perform the Initial setting of the descriptor. | |
|  | Set the channel information to the descriptor. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_find\_channel

|  |  |  |
| --- | --- | --- |
| [Overview] | Search the channel of the specified transaction type | |
|  |  | |
| [Function Name] | dma\_find\_channel | |
|  |  | |
| [Calling format] | struct dma\_chan \*dma\_find\_channel(enum dma\_transaction\_type tx\_type); | |
|  |  | |
| [Arguments] | tx\_type | transaction type |
|  |  | |
| [Returns] | not NULL | applicable channel information |
|  | NULL | no applicable |
|  |  | |
| [Feature] | This function searches the transaction type and returns the channel information. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_issue\_pending\_all

|  |  |
| --- | --- |
| [Overview] | Execute the request if execution of the pending request is possible |
|  |  |
| [Function Name] | dma\_issue\_pending\_all |
|  |  |
| [Calling format] | void dma\_issue\_pending\_all(void); |
|  |  |
| [Arguments] | - |
|  |  |
| [Returns] | - |
|  |  |
| [Feature] | Execute the request if execution of the pending request is possible. All the channels are |
|  | applicable. |
|  |  |
| [Remark] | - |
|  |  |

### dma\_release\_channel

|  |  |  |
| --- | --- | --- |
| [Overview] | Release the channel | |
|  |  | |
| [Function Name] | dma\_release\_channel | |
|  |  | |
| [Calling format] | void dma\_release\_channel(struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | chan | channel to release |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Release the channel. | |
|  | The resource allocate to the specified channels is also released. | |
|  | When the target channel is operating, transmission is forced to stop by the stop order. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_run\_dependencies

|  |  |  |
| --- | --- | --- |
| [Overview] | Perform the dependence processing before execution of a target channel | |
|  |  | |
| [Function Name] | dma\_run\_dependencies | |
|  |  | |
| [Calling format] | void dma\_run\_dependencies(struct dma\_async\_tx\_descriptor \*tx); | |
|  |  | |
| [Arguments] | tx | DMA descriptor that held the target channel |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Perform the dependence processing before execution of a target channel. | |
|  |  | |
| [Remark] | - | |
|  |  | |
|  |  | |

### dma\_submit\_error

|  |  |  |
| --- | --- | --- |
| [Overview] | Check the DMA request cookie | |
|  |  | |
| [Function Name] | dma\_submit\_error | |
|  |  | |
| [Calling format] | int dma\_submit\_error(dma\_cookie\_t cookie); | |
|  |  | |
| [Arguments] | cookie | DMA requested cookie. |
|  |  |  |
|  |  | |
| [Returns] | 0 | OK, It’s DMA request cookie. |
|  | Other than 0 | cookie is Error. |
|  |  | |
| [Feature] | Check the cookie of transaction. | |
|  |  | |
|  |  | |
| [Remark] |  | |
|  |  | |

### dma\_sync\_wait

|  |  |  |  |
| --- | --- | --- | --- |
| [Overview] | Perform waiting process for the completion of transmission | | |
|  |  | | |
| [Function Name] | dma\_sync\_wait | | |
|  |  | | |
| [Calling format] | enum dma\_status dma\_sync\_wait(struct dma\_chan \*chan, dma\_cookie\_t cookie) | | |
|  |  | | |
| [Arguments] | chan | channel information | |
|  | cookie | identical number (use for confirmation of the target queue) | |
|  |  | | |
| [Returns] | DMA\_COMPLETE | | synchronous (transfer) completion |
|  | DMA\_ERROR | | timeout error |
|  |  | | |
| [Feature] | Perform waiting process for the completion of transmission. | | |
|  |  | | |
| [Remark] | - | | |
|  |  | | |

### dma\_wait\_for\_async\_tx

|  |  |  |  |
| --- | --- | --- | --- |
| [Overview] | Perform waiting process for the completion of transmission | | |
|  |  | | |
| [Function Name] | dma\_wait\_for\_async\_tx | | |
|  |  | | |
| [Calling format] | enum dma\_status dma\_wait\_for\_async\_tx(struct dma\_async\_tx\_descriptor \*tx) | | |
|  |  | | |
| [Arguments] | tx | DMA descriptor | |
|  |  | | |
| [Returns] | DMA\_COMPLETE | | transfer completion or argument tx is NULL. |
|  | DMA\_ERROR | | timeout error |
|  |  | | |
| [Feature] | Perform waiting process for the completion of transmission after check whether the | | |
|  | descriptor is busy. | | |
|  |  | | |
| [Remark] | - | | |
|  |  | | |

### dmaengine\_get

|  |  |
| --- | --- |
| [Overview] | Enable to use a free channel with the DMA Engine interface |
|  |  |
| [Function Name] | dmaengine\_get |
|  |  |
| [Calling format] | void dmaengine\_get(void) |
|  |  |
| [Arguments] | - |
|  |  |
| [Returns] | - |
|  |  |
| [Feature] | Enable to use a free channel with the DMA Engine interface. |
|  |  |
| [Remark] | Resource release is required for the termination processing of the channel allocated with |
|  | this function. |
|  | You need to call dmaengine\_put() of the same number of times that called this function. |
|  |  |

### dmaengine\_put

|  |  |
| --- | --- |
| [Overview] | Release the channel that was enabled by the call of dmaengine\_get() |
|  |  |
| [Function Name] | dmaengine\_put |
|  |  |
| [Calling format] | void dmaengine\_put(void) |
|  |  |
| [Arguments] | - |
|  |  |
| [Returns] | - |
|  |  |
| [Feature] | Release the channel that was enabled by the call of dmaengine\_get(). |
|  |  |
| [Remark] | You need to call this function of the same number of times that called dmaengne\_get() if you |
|  | want to terminate the use of the target channel. |
|  |  |

### dma\_cap\_clear

|  |  |  |
| --- | --- | --- |
| [Overview] | Clear the DMA transaction type | |
|  |  | |
| [Function Name] | dma\_cap\_clear | |
|  |  | |
| [Calling format] | void dma\_cap\_clear(enum dma\_transaction\_type tx\_type, dma\_cap\_mask\_t mask); | |
|  |  | |
| [Arguments] | tx\_type | transaction type |
|  | mask | mask structure |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Clear the DMA transaction type (target bit) from the argument mask. | |
|  |  | |
| [Remark] | Refer to 4.5.2.2 about dma\_transaction\_type. | |
|  |  | |

### dma\_cap\_set

|  |  |  |
| --- | --- | --- |
| [Overview] | Set the DMA transaction type | |
|  |  | |
| [Function Name] | dma\_cap\_set | |
|  |  | |
| [Calling format] | void dma\_cap\_set(enum dma\_transaction\_type tx\_type, dma\_cap\_mask\_t mask); | |
|  |  | |
| [Arguments] | tx\_type | transaction type |
|  | mask | mask structure. |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Set the DMA transaction type (target bit) to the argument mask. | |
|  |  | |
| [Remark] | Refer to 4.5.2.2 about dma\_transaction\_type. | |
|  |  | |

### dma\_cap\_zero

|  |  |  |
| --- | --- | --- |
| [Overview] | Initialize the DMA transaction type | |
|  |  | |
| [Function Name] | dma\_cap\_zero | |
|  |  | |
| [Calling format] | void dma\_cap\_zero(dma\_cap\_mask\_t mask); | |
|  |  | |
| [Arguments] | mask | The structure which shows the held transaction type |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Initialize the DMA transaction type that the argument mask is held. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### dma\_has\_cap

|  |  |  |
| --- | --- | --- |
| [Overview] | Check whether holds the DMA transaction type | |
|  |  | |
| [Function Name] | dma\_has\_cap | |
|  |  | |
| [Calling format] | int dma\_has\_cap(enum dma\_transaction\_type tx\_type, dma\_cap\_mask\_t mask); | |
|  |  | |
| [Arguments] | tx\_type | transaction type |
|  | mask | mask structure. |
|  |  | |
| [Returns] | 1 | hold |
|  | 0 | not hold |
|  |  | |
| [Feature] | Check whether there is the DMA transaction type which argument mask holds using the | |
|  | DMA transaction type specified by argument tx\_type. | |
|  |  | |
| [Remark] | Refer to 4.5.2.2 about dma\_transaction\_type. | |
|  |  | |

## DMA device interface function

This section explains about the member function of dma\_device (refer to 4.4.2.2) structure.  
However, the explanation is skipped about the interface that this module is not used.

Table 4-2 DMA device interface function (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| **Chapter** | **Function name** | **Description** |
| --- | --- | --- |
| 4.3.1 | (\*device\_alloc\_chan\_resource)() | Allocate the resource. |
| 4.3.2 | (\*device\_free\_chan\_resource)() | Release the resource. |
| 4.3.3 | (\*device\_prep\_dma\_memcpy)() | Perform the preparation of the transmission. |
| - | (\*device\_prep\_dma\_xor)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_xor\_val)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_pq)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_pq\_val)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_memset)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_memset\_sg)() | This function is not used. (no support) |
| - | (\*device\_prep\_dma\_interrupt)() | This function is not used. (no support) |
| 4.3.4 | (\*device\_prep\_slave\_sg)() | Perform the preparation of the transmission using the device. |
| 4.3.5 | (\*device\_prep\_dma\_cyclic)() | Prepare a cyclic dma operation suitable for audio. |
| - | (\*device\_prep\_interleaved\_dma)() | This function is not used. (no support) |
|  | (\*device\_prep\_dma\_imm\_data)() | This function is not used. (no support) |
| 4.3.6 | (\*device\_config)() | Configure the device. |
| - | (\*device\_pause)() | This function is not used. (no support) |
| - | (\*device\_resume)() | This function is not used. (no support) |
| 4.3.7 | (\*device\_terminate\_all)() | Aborts all transfers. |
| 4.3.8 | (\*device\_synchronize)() | Synchronizes the termination of a transfers to the current context. |
| 4.3.9 | (\*device\_tx\_status)() | Get the status of DMA transmission. |
| 4.3.10 | (\*device\_issue\_pending)() | Execute the transaction if execution of the transaction of the specified channel is possible. |

### device\_alloc\_chan\_resources

|  |  |  |  |
| --- | --- | --- | --- |
| [Overview] | Allocate the resource | | |
|  |  | | |
| [Function Name] | device\_alloc\_chan\_resources | | |
|  |  | | |
| [Calling format] | int (\*device\_alloc\_chan\_resources)(struct dma\_chan \*chan); | | |
|  |  | | |
| [Arguments] | chan | channel information | |
|  |  | | |
| [Returns] | Positive Number | | Success (return the allocated descriptor number) |
|  | -EINVAL | | Error (no such device) |
|  | -EBUSY | | Error (device is busy) |
|  | -ENOMEM | | Error (out of memory) |
|  |  | | |
| [Feature] | Allocate the resource for the specified channel. | | |
|  | This function is called if needed from dmaengine\_get() and dma\_request\_channel() | | |
|  |  | | |
| [Remark] | - | | |
|  |  | | |

### device\_free\_chan\_resources

|  |  |  |
| --- | --- | --- |
| [Overview] | Release the resource | |
|  |  | |
| [Function Name] | device\_free\_chan\_resources | |
|  |  | |
| [Calling format] | void (\*device\_free\_chan\_resources)(struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | chan | channel information |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Release the resource of the specified channel information. | |
|  | The specified channel is forced to stop. | |
|  | This function is called if needed from dmaengine\_put() and dma\_release\_channel(). | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_prep\_dma\_memcpy

|  |  |  |
| --- | --- | --- |
| [Overview] | Perform the preparation of the transmission | |
|  |  | |
| [Function Name] | device\_prep\_dma\_memcpy | |
|  |  | |
| [Calling format] | struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_memcpy)(struct dma\_chan \*chan, | |
|  | dma\_addr\_t dest, dma\_addr\_t src, size\_t len, unsigned long flags); | |
|  |  | |
| [Arguments] | chan | channel information |
|  | dest | destination address (virtual address) |
|  | src | source address (virtual address) |
|  | len | transfer byte size |
|  | flags | DMA control flag (refer to 4.5.2.3) |
|  |  | |
| [Returns] | not NULL | Success |
|  | NULL | Error (chan is invalid, or len is 0) |
|  |  | |
| [Feature] | Perform the setting of address etc. as the preparation of the transmission. | |
|  | This function is called if needed from dma\_async\_memcpy\_buf\_to\_buf(), | |
|  | dma\_async\_memcpy\_buf\_to\_pg(), and dma\_async\_memcpy\_pg\_to\_pg(). | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_prep\_slave\_sg

|  |  |  |
| --- | --- | --- |
| [Overview] | Perform the preparation of the transmission using the device | |
|  |  | |
| [Function Name] | device\_prep\_slave\_sg | |
|  |  | |
| [Calling format] | struct dma\_async\_tx\_descriptor \*(\*device\_prep\_slave\_sg)(struct dma\_chan \*chan, | |
|  | struct scatterlist \*sgl, unsigned int sg\_len, | |
|  | enum dma\_transfer\_direction direction, unsigned long flag, void \*context); | |
|  |  | |
| [Arguments] | chan | channel information |
|  | sgl | SG list |
|  | sg\_len | length of SG list |
|  | direction | direction of transfer |
|  |  | DMA\_TO\_DEVICE : transfer from memory to device |
|  |  | DMA\_FROM\_DEVICE : transfer from device to memory |
|  | flags | DMA control flag (refer to 4.5.2.3) |
|  | context | use for RapidIO, parameter use only NULL on this driver. |
|  |  | |
| [Returns] | not NULL | Success (return the address of the descriptor) |
|  | NULL | Error |
|  |  | |
| [Feature] | Perform the setting of address etc. as the preparation of the transmission using the device. | |
|  | Set up the slave information (struct\_dmae\_slave) to the member private of channel | |
|  | information (struct dma\_chan), and call this function. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_prep\_dma\_cyclic

|  |  |  |
| --- | --- | --- |
| [Overview] | Prepare a cyclic dma operation suitable for audio | |
|  |  | |
| [Function Name] | device\_prep\_dma\_cyclic | |
|  |  | |
| [Calling format] | struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_cyclic)( | |
|  | struct dma\_chan \*chan, dma\_addr\_t buf\_addr, size\_t buf\_len, | |
|  | size\_t period\_len, enum dma\_transfer\_direction direction, | |
|  | unsigned long flags); | |
|  |  | |
| [Arguments] | chan | channel information |
|  | buf\_addr | transfer data buffer address. |
|  | buf\_len | transfer data buffer max length. |
|  | period\_len | cycle transfer period size. the set value need smaller than buf\_len. |
|  | direction | direction of transfer |
|  |  | DMA\_TO\_DEVICE : transfer from memory to device |
|  |  | DMA\_FROM\_DEVICE : transfer from device to memory |
|  | flags | DMA control flag (refer to 4.5.2.3) |
|  |  | |
| [Returns] | not NULL | Success (return the address of the descriptor) |
|  | NULL | Error |
|  |  | |
| [Feature] | Prepare a cyclic dma operation suitable for audio. The function takes a buffer of size buf\_len. | |
|  | The callback function will be called after period\_len bytes have been transferred. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_config

|  |  |  |
| --- | --- | --- |
| [Overview] | Configure the device | |
|  |  | |
| [Function Name] | device\_config | |
|  |  | |
| [Calling format] | int (\*device\_config)(struct dma\_chan \*chan, struct dma\_slave\_config \*config); | |
|  |  | |
| [Arguments] | chan | channel information |
|  | config | config information |
|  |  | |
| [Returns] | 0 | Success |
|  |  | |
| [Feature] | Configure the device. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_terminate\_all

|  |  |  |
| --- | --- | --- |
| [Overview] | Aborts all transfers | |
|  |  | |
| [Function Name] | device\_terminate\_all | |
|  |  | |
| [Calling format] | int (\*device\_terminate\_all)(struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | chan | channel information |
|  |  | |
| [Returns] | 0 | Success |
|  |  | |
| [Feature] | Aborts all transfers. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_synchronize

|  |  |  |
| --- | --- | --- |
| [Overview] | Synchronizes the termination of a transfers to the current context. | |
|  |  | |
| [Function Name] | device\_synchronize | |
|  |  | |
| [Calling format] | void (\*device\_synchronize)(struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | chan | channel information |
|  |  | |
| [Returns] | - | - |
|  |  | |
| [Feature] | Synchronizes to the DMA channel termination to the current context. When this function | |
|  | returns it is guaranteed that all transfers for previously issued descriptors have stopped and | |
|  | and it is safe to free the memory assoicated with them. Furthermore it is guaranteed that all | |
|  | complete callback functions for a previously submitted descriptor have finished running and it | |
|  | is safe to free resources accessed from within the complete callbacks. | |
|  |  | |
| [Remark] | - | |
|  |  | |

### device\_tx\_status

|  |  |  |  |
| --- | --- | --- | --- |
| [Overview] | Get the status of DMA transmission | | |
|  |  | | |
| [Function Name] | device\_tx\_status | | |
|  |  | | |
| [Calling format] | enum dma\_status (\*device\_tx\_status)(struct dma\_chan \*chan, dma\_cookie\_t cookie, | | |
|  | struct dma\_tx\_state \*txstate); | | |
|  |  | | |
| [Arguments] | chan | channel information | |
|  | cookie | identical information | |
|  | txstate | Specify the area which stores the status of DMA | |
|  |  | | |
| [Returns] | DMA\_COMPLETE | | transaction completed successfully |
|  | DMA\_IN\_PROGRESS | | transaction not yet processed |
|  | DMA\_ERROR | | transaction failed |
|  |  | | |
| [Feature] | Get the status of DMA transmission. | | |
|  |  | | |
| [Remark] | Refer to 4.4.2.6 about dma\_tx\_state structure. | | |
|  |  | | |

### device\_issue\_pending

|  |  |  |
| --- | --- | --- |
| [Overview] | Execute the transaction if execution of the transaction of the specified channel is possible | |
|  |  | |
| [Function Name] | device\_issue\_pending | |
|  |  | |
| [Calling format] | void (\*device\_issue\_pending)(struct dma\_chan \*chan); | |
|  |  | |
| [Arguments] | chan | channel information |
|  |  | |
| [Returns] | - | |
|  |  | |
| [Feature] | Execute the transaction if execution of the transaction of the specified channel is possible. | |
|  | That is, when the specified channel is during transmission, this function returns without | |
|  | performing the unperformed transaction. | |
|  |  | |
| [Remark] | - | |
|  |  | |

## Structure

### This module structures

This section shows the structure that this module is defined.

#### rcar\_dmac\_transfer\_chunk

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_xfer\_chunk - Descriptor for a hardware transfer

\* @node: entry in the parent's chunks list

\* @src\_addr: device source address

\* @dst\_addr: device destination address

\* @size: transfer size in bytes

\*/

struct rcar\_dmac\_xfer\_chunk {

struct list\_head node;

dma\_addr\_t src\_addr;

dma\_addr\_t dst\_addr;

u32 size;

};

**Figure 4-1 struct rcar\_dmac\_xfer\_chunk (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_hw\_desc

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_hw\_desc - Hardware descriptor for a transfer chunk

\* @sar: value of the SAR register (source address)

\* @dar: value of the DAR register (destination address)

\* @tcr: value of the TCR register (transfer count)

\*/

struct rcar\_dmac\_hw\_desc {

u32 sar;

u32 dar;

u32 tcr;

u32 reserved;

} \_\_attribute\_\_((\_\_packed\_\_));

**Figure 4-2 struct rcar\_dmac\_hw\_desc (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_desc\_page

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_desc\_page - One page worth of descriptors

\* @node: entry in the channel's pages list

\* @descs: array of DMA descriptors

\* @chunks: array of transfer chunk descriptors

\*/

struct rcar\_dmac\_desc\_page {

struct list\_head node;

union {

struct rcar\_dmac\_desc descs[0];

struct rcar\_dmac\_xfer\_chunk chunks[0];

};

};

**Figure 4-3 struct rcar\_dmac\_desc\_page (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_chan\_slave

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_chan\_slave - Slave configuration

\* @slave\_addr: slave memory address

\* @xfer\_size: size (in bytes) of hardware transfers

\*/

struct rcar\_dmac\_chan\_slave {

phys\_addr\_t slave\_addr;

unsigned int xfer\_size;

};

**Figure 4-4 struct rcar\_dmac\_chan\_slave (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_chan\_map

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_chan\_map - Map of slave device phys to dma address

\* @addr: slave dma address

\* @dir: direction of mapping

\* @slave: slave configuration that is mapped

\*/

struct rcar\_dmac\_chan\_map {

dma\_addr\_t addr;

enum dma\_data\_direction dir;

struct rcar\_dmac\_chan\_slave slave;

};

**Figure 4-5 struct rcar\_dmac\_chan\_map (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_chan

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac\_chan - R-Car Gen2 DMA Controller Channel

\* @chan: base DMA channel object

\* @iomem: channel I/O memory base

\* @index: index of this channel in the controller

\* @irq: channel IRQ

\* @src: slave memory address and size on the source side

\* @dst: slave memory address and size on the destination side

\* @mid\_rid: hardware MID/RID for the DMA client using this channel

\* @lock: protects the channel CHCR register and the desc members

\* @desc.free: list of free descriptors

\* @desc.pending: list of pending descriptors (submitted with tx\_submit)

\* @desc.active: list of active descriptors (activated with issue\_pending)

\* @desc.done: list of completed descriptors

\* @desc.wait: list of descriptors waiting for an ack

\* @desc.running: the descriptor being processed (a member of the active list)

\* @desc.chunks\_free: list of free transfer chunk descriptors

\* @desc.pages: list of pages used by allocated descriptors

\*/

struct rcar\_dmac\_chan {

struct dma\_chan chan;

void \_\_iomem \*iomem;

unsigned int index;

int irq;

struct rcar\_dmac\_chan\_slave src;

struct rcar\_dmac\_chan\_slave dst;

struct rcar\_dmac\_chan\_map map;

int mid\_rid;

spinlock\_t lock;

struct {

struct list\_head free;

struct list\_head pending;

struct list\_head active;

struct list\_head done;

struct list\_head wait;

struct rcar\_dmac\_desc \*running;

struct list\_head chunks\_free;

struct list\_head pages;

} desc;

};

**Figure 4-6 struct rcar\_dmac\_chan (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac

This structure is defined in drivers/dma/sh/rcar-dmac.c.

/\*

\* struct rcar\_dmac - R-Car Gen2 DMA Controller

\* @engine: base DMA engine object

\* @dev: the hardware device

\* @dmac\_base: remapped base register block

\* @chan\_base: remapped channel register block (optional)

\* @n\_channels: number of available channels

\* @channels: array of DMAC channels

\* @fixed\_source: fixed source address mode

\* @fixed\_dest: fixed destination address mode

\* @rate\_rd: bus read rate control

\* @rate\_wr: bus write rate control

\* @modules: bitmask of client modules in use

\*/

struct rcar\_dmac {

struct dma\_device engine;

struct device \*dev;

void \_\_iomem \*dmac\_base;

void \_\_iomem \*chan\_base;

struct device\_dma\_parameters parms;

unsigned int n\_channels;

struct rcar\_dmac\_chan \*channels;

u32 channels\_mask;

bool fixed\_source;

bool fixed\_dest;

unsigned int rate\_rd;

unsigned int rate\_wr;

DECLARE\_BITMAP(modules, 256);

};

**Figure 4-7 struct rcar\_dmac (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### rcar\_dmac\_of\_data

This structure is defined in drivers/dma/sh/rcar-dmac.c

/\*

\* struct rcar\_dmac\_of\_data - This driver's OF data

\* @chan\_offset\_base: DMAC channels base offset

\* @chan\_offset\_stride: DMAC channels offset stride

\*/

struct rcar\_dmac\_of\_data {

u32 chan\_offset\_base;

u32 chan\_offset\_stride;

};

Figure 4-8 struct rcar\_dmac\_of\_data (R-Car H3/M3/M3N/E3/V3U/V3H)

#### usb\_dmac\_sg

This structure is defined in drivers/dma/sh/usb-dmac.c.

/\*

\* struct usb\_dmac\_sg - Descriptor for a hardware transfer

\* @mem\_addr: memory address

\* @size: transfer size in bytes

\*/

struct usb\_dmac\_sg {

dma\_addr\_t mem\_addr;

u32 size;

};

**Figure 4-9 struct usb\_dmac\_sg (R-Car H3/M3/M3N/E3/D3)**

#### usb\_dmac\_desc

This structure is defined in drivers/dma/sh/usb-dmac.c.

/\*

\* struct usb\_dmac\_desc - USB DMA Transfer Descriptor

\* @vd: base virtual channel DMA transaction descriptor

\* @direction: direction of the DMA transfer

\* @sg\_allocated\_len: length of allocated sg

\* @sg\_len: length of sg

\* @sg\_index: index of sg

\* @residue: residue after the DMAC completed a transfer

\* @node: node for desc\_got and desc\_freed

\* @done\_cookie: cookie after the DMAC completed a transfer

\* @sg: information for the transfer

\*/

struct usb\_dmac\_desc {

struct virt\_dma\_desc vd;

enum dma\_transfer\_direction direction;

unsigned int sg\_allocated\_len;

unsigned int sg\_len;

unsigned int sg\_index;

u32 residue;

struct list\_head node;

dma\_cookie\_t done\_cookie;

struct usb\_dmac\_sg sg[0];

};

**Figure 4-10 struct usb\_dmac\_desc (R-Car H3/M3/M3N/E3/D3)**

#### usb\_dmac\_chan

This structure is defined in drivers/dma/sh/usb-dmac.c.

/\*

\* struct usb\_dmac\_chan - USB DMA Controller Channel

\* @vc: base virtual DMA channel object

\* @iomem: channel I/O memory base

\* @index: index of this channel in the controller

\* @irq: irq number of this channel

\* @desc: the current descriptor

\* @descs\_allocated: number of descriptors allocated

\* @desc\_got: got descriptors

\* @desc\_freed: freed descriptors after the DMAC completed a transfer

\*/

struct usb\_dmac\_chan {

struct virt\_dma\_chan vc;

void \_\_iomem \*iomem;

unsigned int index;

int irq;

struct usb\_dmac\_desc \*desc;

int descs\_allocated;

struct list\_head desc\_got;

struct list\_head desc\_freed;

};

**Figure 4-11 struct usb\_dmac\_chan (R-Car H3/M3/M3N/E3/D3)**

#### usb\_dmac

This structure is defined in drivers/dma/sh/usb-dmac.c.

/\*

\* struct usb\_dmac - USB DMA Controller

\* @engine: base DMA engine object

\* @dev: the hardware device

\* @iomem: remapped I/O memory base

\* @n\_channels: number of available channels

\* @channels: array of DMAC channels

\*/

struct usb\_dmac {

struct dma\_device engine;

struct device \*dev;

void \_\_iomem \*iomem;

unsigned int n\_channels;

struct usb\_dmac\_chan \*channels;

};

**Figure 4-12 struct usb\_dmac (R-Car H3/M3/M3N/E3/D3)**

### DMA Engine Framework structure

This section shows the DMA Engine Framework structure that this module is used. These structure is defined in include/linux/dmaengine.h.

#### dma\_chan

struct dma\_chan {

struct dma\_device \*device;

dma\_cookie\_t cookie;

dma\_cookie\_t completed\_cookie;

/\* sysfs \*/

int chan\_id;

struct dma\_chan\_dev \*dev;

struct list\_head device\_node;

struct dma\_chan\_percpu \_\_percpu \*local;

int client\_count;

int table\_count;

/\* DMA router \*/

struct dma\_router \*router;

void \*route\_data;

void \*private;

};

**Figure 4-13 struct dma\_chan (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_device

struct dma\_device {

unsigned int chancnt;

unsigned int privatecnt;

struct list\_head channels;

struct list\_head global\_node;

struct dma\_filter filter;

dma\_cap\_mask\_t cap\_mask;

enum dma\_desc\_metadata\_mode desc\_metadata\_modes;

unsigned short max\_xor;

unsigned short max\_pq;

enum dmaengine\_alignment copy\_align;

enum dmaengine\_alignment xor\_align;

enum dmaengine\_alignment pq\_align;

enum dmaengine\_alignment fill\_align;

#define DMA\_HAS\_PQ\_CONTINUE (1 << 15)

int dev\_id;

struct device \*dev;

struct module \*owner;

struct ida chan\_ida;

struct mutex chan\_mutex; /\* to protect chan\_ida \*/

u32 src\_addr\_widths;

u32 dst\_addr\_widths;

u32 directions;

u32 min\_burst;

u32 max\_burst;

bool descriptor\_reuse;

enum dma\_residue\_granularity residue\_granularity;

int (\*device\_alloc\_chan\_resources)(struct dma\_chan \*chan);

void (\*device\_free\_chan\_resources)(struct dma\_chan \*chan);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_memcpy)(

struct dma\_chan \*chan, dma\_addr\_t dst, dma\_addr\_t src,

size\_t len, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_xor)(

struct dma\_chan \*chan, dma\_addr\_t dst, dma\_addr\_t \*src,

unsigned int src\_cnt, size\_t len, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_xor\_val)(

struct dma\_chan \*chan, dma\_addr\_t \*src, unsigned int src\_cnt,

size\_t len, enum sum\_check\_flags \*result, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_pq)(

struct dma\_chan \*chan, dma\_addr\_t \*dst, dma\_addr\_t \*src,

unsigned int src\_cnt, const unsigned char \*scf,

size\_t len, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_pq\_val)(

struct dma\_chan \*chan, dma\_addr\_t \*pq, dma\_addr\_t \*src,

unsigned int src\_cnt, const unsigned char \*scf, size\_t len,

enum sum\_check\_flags \*pqres, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_memset)(

struct dma\_chan \*chan, dma\_addr\_t dest, int value, size\_t len,

unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_memset\_sg)(

struct dma\_chan \*chan, struct scatterlist \*sg,

unsigned int nents, int value, unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_interrupt)(

struct dma\_chan \*chan, unsigned long flags);

**Figure 4-14 struct dma\_device (R-Car H3/M3/M3N/E3/D3/V3U/V3H) (1/2)**

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_slave\_sg)(

struct dma\_chan \*chan, struct scatterlist \*sgl,

unsigned int sg\_len, enum dma\_transfer\_direction direction,

unsigned long flags, void \*context);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_cyclic)(

struct dma\_chan \*chan, dma\_addr\_t buf\_addr, size\_t buf\_len,

size\_t period\_len, enum dma\_transfer\_direction direction,

unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_interleaved\_dma)(

struct dma\_chan \*chan, struct dma\_interleaved\_template \*xt,

unsigned long flags);

struct dma\_async\_tx\_descriptor \*(\*device\_prep\_dma\_imm\_data)(

struct dma\_chan \*chan, dma\_addr\_t dst, u64 data,

unsigned long flags);

void (\*device\_caps)(struct dma\_chan \*chan,

struct dma\_slave\_caps \*caps);

int (\*device\_config)(struct dma\_chan \*chan,

struct dma\_slave\_config \*config);

int (\*device\_pause)(struct dma\_chan \*chan);

int (\*device\_resume)(struct dma\_chan \*chan);

int (\*device\_terminate\_all)(struct dma\_chan \*chan);

void (\*device\_synchronize)(struct dma\_chan \*chan);

enum dma\_status (\*device\_tx\_status)(struct dma\_chan \*chan,

dma\_cookie\_t cookie,

struct dma\_tx\_state \*txstate);

void (\*device\_issue\_pending)(struct dma\_chan \*chan);

void (\*device\_release)(struct dma\_device \*dev);

};

**Figure 4-15 struct dma\_device (R-Car H3/M3/M3N/E3/D3/V3U/V3H) (2/2)**

#### dma\_cap\_mask\_t

transaction\_type

typedef struct { DECLARE\_BITMAP(bits, DMA\_TX\_TYPE\_END); } dma\_cap\_mask\_t;

**Figure 4-16 struct dma\_cap\_mask\_t (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_async\_tx\_descriptor

struct dma\_async\_tx\_descriptor {

dma\_cookie\_t cookie;

enum dma\_ctrl\_flags flags; /\* not a 'long' to pack with cookie \*/

dma\_addr\_t phys;

struct dma\_chan \*chan;

dma\_cookie\_t (\*tx\_submit)(struct dma\_async\_tx\_descriptor \*tx);

int (\*desc\_free)(struct dma\_async\_tx\_descriptor \*tx);

dma\_async\_tx\_callback callback;

dma\_async\_tx\_callback\_result callback\_result;

void \*callback\_param;

strunct dmaengine\_unmap\_data \*unmap;

#ifdef CONFIG\_ASYNC\_TX\_ENABLE\_CHANNEL\_SWITCH

struct dma\_async\_tx\_descriptor \*next;

struct dma\_async\_tx\_descriptor \*parent;

spinlock\_t lock;

#endif

};

**Figure 4-17 struct dma\_async\_tx\_descriptor (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dmaengine\_unmap\_data

struct dmaengine\_unmap\_data {

u8 map\_cnt;

u8 to\_cnt;

u8 from\_cnt;

u8 bidi\_cnt;

struct device \*dev;

struct kref kref;

size\_t len;

dma\_addr\_t addr[];

};

**Figure 4-18 struct dmaengine\_unmap\_data (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_tx\_state

struct dma\_tx\_state {

dma\_cookie\_t last;

dma\_cookie\_t used;

u32 residue;

u32 in\_flight\_bytes;

};

**Figure 4-19 struct dma\_tx\_state (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_chan\_dev

struct dma\_chan\_dev {

struct dma\_chan \*chan;

struct device device;

int dev\_id;

};

**Figure 4-20 struct dma\_chan\_dev (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_chan\_percpu

struct dma\_chan\_percpu {

/\* stats \*/

unsigned long memcpy\_count;

unsigned long bytes\_transferred;

};

**Figure 4-81 struct dma\_chan\_percpu (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_router

struct dma\_router {

struct device \*dev;

void (\*route\_free)(struct device \*dev, void \*route\_data);

};

**Figure 4-92 struct dma\_router (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

#### dma\_slave\_config

struct dma\_slave\_config {

enum dma\_transfer\_direction direction;

phys\_addr\_t src\_addr;

phys\_addr\_t dst\_addr;

enum dma\_slave\_buswidth src\_addr\_width;

enum dma\_slave\_buswidth dst\_addr\_width;

u32 src\_maxburst;

u32 dst\_maxburst;

u32 src\_port\_window\_size;

u32 dst\_port\_window\_size;

bool device\_fc;

unsigned int slave\_id;

};

**Figure 4-103 struct dma\_slave\_config (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

## Global Variables and Constants

### Global Variables

There are no global variables for this module.

### Global Constants

This section shows global constants that this module is used.

#### dma\_status

Table 4-3 List of DMA status definition (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| **Definition Name** | Status |
| --- | --- |
| DMA\_COMPLETE | Transaction completed successfully |
| DMA\_IN\_PROGRESS | Transaction not yet processed |
| DMA\_PAUSED | Transaction is paused |
| DMA\_ERROR | Transaction failed |

#### dma\_transaction\_type

The transaction type is not set in this module, this is set when the device that is used this module is registered.

Table 4-4 DMA transaction type (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| **Definition Name** | Support |
| --- | --- |
| DMA\_MEMCPY | This is support in this module. |
| DMA\_XOR | This is no support in this module. |
| DMA\_PQ | This is no support in this module. |
| DMA\_XOR\_VAL | This is no support in this module. |
| DMA\_PQ\_VAL | This is no support in this module. |
| DMA\_MEMSET | This is no support in this module. |
| DMA\_MEMSET\_SG | This is no support in this module. |
| DMA\_INTERRUPT | This is no support in this module. |
| DMA\_PRIVATE | This is support in this module. |
| DMA\_ASYNC\_TX | This is support in this module. |
| DMA\_SLAVE | This is support in this module. |
| DMA\_CYCLIC | This is no support in this module. |
| DMA\_INTERLEAVE | This is no support in this module. |

#### dma\_ctrl\_flags

Table 4-5 List of DMA control flag (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

| **Definition Name** | Explanation |
| --- | --- |
| DMA\_PREP\_INTERRUPT | Trigger an interrupt in completion. |
| DMA\_CTRL\_ACK | Control that descriptor reuse is prohibited until receiving the operation result. |
| DMA\_PREP\_PQ\_DISABLE\_P | Prevent generation of P while generating Q. |
| DMA\_PREP\_PQ\_DISABLE\_Q | Prevent generation of Q while generating P. |
| DMA\_PREP\_CONTINUE | Reuse the source. |
| DMA\_PREP\_FENCE | Tell the driver that subsequent operations depend on the result of this operation. |
| DMA\_CTRL\_REUSE | Client can reuse the descriptor and submit again till cleared or freed. |
| DMA\_PREP\_CMD | Tell the driver that the data passed to DMA API is command data and the descriptor should be in different format from normal data descriptors. |

## Example of use

This section explains the process sequence about the transmission between the memory and the peripheral.

### DMA property

DMA Engine driver reads property from DT.

Required properties:

- dmas:

a list of <[DMA multiplexer phandle] [MID/RID value]> pairs, where MID/RID values are fixed handles, specified in the SoC manual.

- dma-names:

a list of DMA channel names, one per "dmas" entry

Example: please see **Figure 4-11**4

dmas = <&dmac1 0x95>, <&dmac1 0x94>,

<&dmac2 0x95>, <&dmac2 0x94>;

dma-names = "tx", "rx", "tx", "rx";

**Figure 4-114 Example of DMA property (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Getting channel

If you specify an association with a device in the device tree file, use the dma\_request\_chan() to acquire the channel. If no association is specified, the channel is acquired using the dma\_request\_channel() as follows.

The argument passed to dma\_request\_channel() is set up in advance.  
A variable mask (dma\_cap\_mask\_t structure) is initialized in dma\_cap\_zero(), "DMA\_SLAVE" is set to mask in dma\_cap\_zero().

dma\_cap\_mask\_t mask;

dma\_cap\_zero(mask);

dma\_cap\_set(DMA\_SLAVE, mask);

**Figure 4-125 Getting channel (1) (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

The filter function of the 2nd argument of a dma\_request\_channel() is created according to the following forms.  
bool (\*dma\_filter\_fn)(struct dma\_chan \*chan, void \*filter\_param);

It enables you to obtain the channel that meets the conditions that you review the acquisition channels with this filter function to return a true / false. If it returns true, it will use as an available channel. If it returns false, this filter function is called again for find the channel available to new.

The following, in case there is no determination of conditions in the filter function, it is the example to be used as the channel that it finds.

static bool filter(struct dma\_chan \*chan, void \*arg)

{

chan->private = arg;

return true;

}

**Figure 4-136 Example of filter function (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

The 3rd argument of dma\_request\_channel() is passed as parameter to the filter function.  
Set the slave configuration information as the private member of channel information structure (struct dma\_chan), and pass it to this module.

struct of\_phandle\_args \*dma\_spec

/\* Set the information to of\_dma structure \*/

chan = dma\_request\_channel(mask, filter, &dma\_spec);

**Figure 4-147 Getting channel (2) (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Preparation of transmission

In transmission of the device, transmission preparation of the setting of the transmission address etc. is performed using SG list (scatterlist structure). It sets up in transmission preparation function device\_prep\_slave\_sg().  
Set "DMA\_FROM\_DEVICE" as the argument at the time of the transmission to the memory from the peripheral, and set "DMA\_TO\_DEVICE" as the argument at the time of the transmission to the peripheral from the memory.

/\* In the example shown below, the transfer area shall assume that the area

allocated as the non-caching area is used, and address buf\_addr and size buf\_size

shall be separately passed by the argument etc. \*/

struct scatterlist sg;

struct dma\_async\_tx\_descriptor \*desc = NULL;

unsigned int sg\_len;

sg\_len = 1;

sg\_init\_table(&sg, sg\_len);

sg\_set\_page(&sg, pfn\_to\_page(PFN\_DOWN(buff\_addr)), /\* Set the offset of the SG list \*/

buf\_size, offset\_in\_page(buff\_addr)); /\* and the size \*/

sg\_dma\_address(&sg) = buf\_addr; /\* Set the address to the SG list \*/

desc = chan->device->device\_prep\_slave\_sg(chan, &sg, sg\_len,

DMA\_TO\_DEVICE, DMA\_PREP\_INTERRUPT | DMA\_CTRL\_ACK);

**Figure 4-158 Preparation of transmission (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Request DMA transfer

Since the completion interruption of transmission was validated 4.6.2, a callback function and a parameter are set, and the transmission request function tx\_submit() is called.  
dmaengine\_submit() is the macro which calls tx\_submit().

struct completion cmp;

dma\_cookie\_t cookie;

init\_completion(&cmp);

desc->callback = dma\_callback;

desc->callback\_param = &cmp;

cookie = desc->tx\_submit(desc);

**Figure 4-169 Request DMA transfer (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Execute DMA transfer

Execute the transfer of the specified channel.

chan->device->device\_issue\_pending(chan);

**Figure 4-30 Execute DMA transfer (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Callback process of transmission completion

Transmission completion will call a callback function. Perform process after transmission if needed.

static void dma\_callback(void \*completion) {

complete(completion);

}

**Figure 4-171 Callback process of transmission completion (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Confirmation of transmission completion

The transmission completion is confirmed.

unsigned long tmo = msecs\_to\_jiffies(3000);

tmo = wait\_for\_completion\_timeout(&cmp, tmo);

status = dma\_async\_is\_tx\_complete(chan, cookie, NULL, NULL);

**Figure 4-182 Confirmation of transmission completion (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

### Termination

Release the allocated channel, when you complete transmission.

dma\_release\_channel(chan);

**Figure 4-193 Termination (R-Car H3/M3/M3N/E3/D3/V3U/V3H)**

# Integration

## Directory Configuration

The directory configuration is shown below.

drivers/dma/sh

usb-dmac.c

：USBHS-DMAC driver source file

rcar-dmac.c

：SYS-DMAC/Audio-DMAC/RT-DMAC driver source file

Figure 5-1 Directory configuration (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

## Integration Procedure

To enable the function of this module, make the following setting with Kernel Configuration.

Device Drivers --->

[\*] DMA Engine support --->

[ ] Renesas SuperH DMA Engine support

<\*> Renesas R-Car Gen2 DMA Controller

<\*> Renesas USB-DMA　Controller

Figure 5-2 Kernel configuration (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

## Device Tree Setting

Basic configuration information of Audio-DMAC is defined at device tree file (r8a7795-es1.dtsi, r8a7795.dtsi,　r8a7796.dtsi, r8a77965.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

The initial reference information is shown below.

SYS-DMAC, USBHS-DMAC, and RT-DMAC are much the same configuration as Audio-DMAC (address and property details are different).

audma0: dma-controller@ec700000 {

compatible = "renesas,dmac-r8a7795", // … \*1

"renesas,rcar-dmac";

reg = <0 0xec700000 0 0x10000>;

interrupts = <GIC\_SPI 350 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 320 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 321 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 322 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 323 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 324 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 325 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 326 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 327 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 328 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 329 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 330 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 331 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 332 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 333 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 334 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 335 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 502>;

clock-names = "fck";

power-domains = <&sysc R8A7795\_PD\_ALWAYS\_ON>; // … \*2

resets = <&cpg 502>;

#dma-cells = <1>;

dma-channels = <16>;

iommus = <&ipmmu\_mp0 0>, <&ipmmu\_mp0 1>,

<&ipmmu\_mp0 2>, <&ipmmu\_mp0 3>,

<&ipmmu\_mp0 4>, <&ipmmu\_mp0 5>,

<&ipmmu\_mp0 6>, <&ipmmu\_mp0 7>,

<&ipmmu\_mp0 8>, <&ipmmu\_mp0 9>,

<&ipmmu\_mp0 10>, <&ipmmu\_mp0 11>,

<&ipmmu\_mp0 12>, <&ipmmu\_mp0 13>,

<&ipmmu\_mp0 14>, <&ipmmu\_mp0 15>;

};

\*1: “renesas,dmac-r8a7795” is for R-Car H3. “renesas,dmac-r8a7796” is for R-Car M3. “renesas,dmac-r8a77965” is for R-Car M3N. "renesas,dmac-r8a77990" is for R-Car E3. "renesas,dmac-r8a77995" is for R-Car D3.

\*2: “R8A7795\_PD\_ALWAYS\_ON” is for R-Car H3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3. “R8A77965\_PD\_ALWAYS\_ON” is for R-Car M3N. “R8A77990\_PD\_ALWAYS\_ON” is for R-Car E3. “R8A77995\_PD\_ALWAYS\_ON” is for R-Car D3.

Figure 5-3 Device tree initial references information of Audio-DMAC0 (R-Car H3/M3/M3N/E3/D3)

audma1: dma-controller@ec720000 {

compatible = "renesas,dmac-r8a7795", // … \*1

"renesas,rcar-dmac";

reg = <0 0xec720000 0 0x10000>;

interrupts = <GIC\_SPI 351 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 336 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 337 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 338 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 339 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 340 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 341 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 342 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 343 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 344 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 345 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 346 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 347 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 348 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 349 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 382 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 383 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 501>;

clock-names = "fck";

power-domains = <&sysc R8A7795\_PD\_ALWAYS\_ON>; // … \*2

resets = <&cpg 502>;

#dma-cells = <1>;

dma-channels = <16>;

iommus = <&ipmmu\_mp0 16>, <&ipmmu\_mp0 17>,

<&ipmmu\_mp0 18>, <&ipmmu\_mp0 19>,

<&ipmmu\_mp0 20>, <&ipmmu\_mp0 21>,

<&ipmmu\_mp0 22>, <&ipmmu\_mp0 23>,

<&ipmmu\_mp0 24>, <&ipmmu\_mp0 25>,

<&ipmmu\_mp0 26>, <&ipmmu\_mp0 27>,

<&ipmmu\_mp0 28>, <&ipmmu\_mp0 29>,

<&ipmmu\_mp0 30>, <&ipmmu\_mp0 31>;

};

\*1: “renesas,dmac-r8a7795” is for R-Car H3. “renesas,dmac-r8a7796” is for R-Car M3. “renesas,dmac-r8a77965” is for R-Car M3N.

\*2: “R8A7795\_PD\_ALWAYS\_ON” is for R-Car H3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3N.

Figure 5-4 Device tree initial references information of Audio-DMAC1 (R-Car H3/M3/M3N)

|  |
| --- |
| dmac0: dma-controller@e6700000 {  compatible = "renesas,dmac-r8a77990",  "renesas,rcar-dmac";  reg = <0 0xe6700000 0 0x10000>;  interrupts = <GIC\_SPI 199 IRQ\_TYPE\_LEVEL\_HIGH,  GIC\_SPI 200 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 201 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 202 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 203 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 204 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 205 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 206 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 207 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 208 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 209 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 210 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 211 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 212 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 213 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 214 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 215 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 219>;  clock-names = "fck";  power-domains = <&sysc R8A77990\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-5 Device tree initial references information of SYS-DMAC0 (R-Car E3)

|  |
| --- |
| dmac1: dma-controller@e7300000 {  compatible = "renesas,dmac-r8a77990",  "renesas,rcar-dmac";  reg = <0 0xe7300000 0 0x10000>;  interrupts = <GIC\_SPI 220 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 216 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 217 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 218 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 219 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 308 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 309 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 310 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 311 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 312 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 313 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 314 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 315 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 316 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 317 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 318 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 319 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 218>;  clock-names = "fck";  power-domains = <&sysc R8A77990\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-6 Device tree initial references information of SYS-DMAC1 (R-Car E3)

|  |
| --- |
| dmac2: dma-controller@e7310000 {  compatible = "renesas,dmac-r8a77990",  "renesas,rcar-dmac";  reg = <0 0xe7310000 0 0x10000>;  interrupts = < GIC\_SPI 416 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 417 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 418 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 419 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 420 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 421 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 422 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 423 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 424 IRQ\_TYPE\_LEVEL\_HIGH>;  GIC\_SPI 425 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 426 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 427 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 428 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 429 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 430 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 431 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 397 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 217>;  clock-names = "fck";  power-domains = <&sysc R8A77990\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-7 Device tree initial references information of SYS-DMAC2 (R-Car E3)

|  |
| --- |
| usb\_dmac0: dma-controller@e65a0000 {  compatible = "renesas,r8a77990-usb-dmac",  "renesas,usb-dmac";  reg = <0 0xe65a0000 0 0x100>;  interrupts = <GIC\_SPI 109 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 109 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "ch0", "ch1";  clocks = <&cpg CPG\_MOD 330>;  power-domains = <&sysc R8A77990\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <2>;  };  usb\_dmac1: dma-controller@e65b0000 {  compatible = "renesas,r8a77990-usb-dmac",  "renesas,usb-dmac";  reg = <0 0xe65b0000 0 0x100>;  interrupts = <GIC\_SPI 110 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 110 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "ch0", "ch1";  clocks = <&cpg CPG\_MOD 331>;  power-domains = <&sysc R8A77990\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <2>;  }; |

Figure 5-8 Device tree initial references information of USB-DMAC (R-Car E3)

Basic configuration information of RT-DMAC should be defined at device tree file (r8a7795-es1.dtsi, r8a7795.dtsi,　r8a7796.dtsi, r8a77965.dtsi, r8a77990.dtsi, r8a77995.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

rt\_dmac0: dma-controller@ffc10000 {

compatible = "renesas,dmac-r8a7795", //… \*1

"renesas,rcar-dmac";

reg = <0 0xffc10000 0 0x10000>;

interrupts = <GIC\_SPI 448 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 449 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 450 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 451 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 452 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 453 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 454 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 455 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 456 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7";

clocks = <&cpg CPG\_MOD 21>;

clock-names = "fck";

power-domains = <&sysc R8A7795\_PD\_ALWAYS\_ON>; // … \*2

resets = <&cpg 21>;

#dma-cells = <1>;

dma-channels = <8>;

};

\*1: “renesas,dmac-r8a7795” is for R-Car H3. “renesas,dmac-r8a7796” is for R-Car M3. “renesas,dmac-r8a77965” is for R-Car M3N. “renesas,dmac-r8a77990” is for R-Car E3.

\*2: “R8A7795\_PD\_ALWAYS\_ON” is for R-Car H3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3N. “R8A77990\_PD\_ALWAYS\_ON” is for R-Car E3.

Figure 5-9 Device tree initial references information of RT-DMAC0 (R-Car H3/M3/M3N/E3)

rt\_dmac1: dma-controller@ffc20000 {

compatible = "renesas,dmac-r8a7795", //… \*1

"renesas,rcar-dmac";

reg = <0 0xffc20000 0 0x10000>;

interrupts = <GIC\_SPI 469 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 457 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 458 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 459 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 460 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 461 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 462 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 463 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 464 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7";

clocks = <&cpg CPG\_MOD 16>;

clock-names = "fck";

power-domains = <&sysc R8A7795\_PD\_ALWAYS\_ON>; // … \*2

resets = <&cpg 16>;

#dma-cells = <1>;

dma-channels = <8>;

};

\*1: “renesas,dmac-r8a7795” is for R-Car H3. “renesas,dmac-r8a7796” is for R-Car M3. “renesas,dmac-r8a77965” is for R-Car M3N. “renesas,dmac-r8a77990” is for R-Car E3. “renesas,dmac-r8a77995” is for R-Car D3.

\*2: “R8A7795\_PD\_ALWAYS\_ON” is for R-Car H3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3. “R8A7796\_PD\_ALWAYS\_ON” is for R-Car M3N. “R8A77990\_PD\_ALWAYS\_ON” is for R-Car E3. “R8A77995\_PD\_ALWAYS\_ON” is for R-Car D3.

Figure 5-10 Device tree initial references information of RT-DMAC1 (R-Car H3/M3N/E3/D3)

Basic configuration information of R-Car V3U SYS-DMAC is defined at device tree file (r8a779a0.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

dmac1: dma-controller@e7350000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-dmac";

reg = <0 0xe7350000 0 0x1000>,

<0 0xe7300000 0 0xf104>;

interrupts = <GIC\_SPI 6 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 32 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 33 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 34 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 35 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 36 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 37 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 38 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 39 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 40 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 41 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 42 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 43 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 44 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 45 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 46 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 47 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 709>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 709>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-11 Device tree initial references information of SYS-DMAC1 (R-Car V3U)

dmac2: dma-controller@e7351000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-dmac";

reg = <0 0xe7351000 0 0x1000>,

<0 0xe7310000 0 0xf104>;

interrupts = <GIC\_SPI 7 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 48 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 49 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 50 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 51 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 52 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 53 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 54 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 55 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7";

clocks = <&cpg CPG\_MOD 710>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 710>;

#dma-cells = <1>;

dma-channels = <8>;

};

Figure 5-12 Device tree initial references information of SYS-DMAC2 (R-Car V3U)

Basic configuration information of R-Car V3U RT-DMAC is defined at device tree file (r8a779a0.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

rt\_dmac0: dma-controller@ffd60000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-rt-dmac";

reg = <0 0xffd60000 0 0x1000>,

<0 0xffc10000 0 0xf104>;

interrupts = <GIC\_SPI 8 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 64 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 65 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 66 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 67 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 68 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 69 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 70 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 71 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 72 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 73 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 74 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 75 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 76 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 77 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 78 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 79 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 630>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 630>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-13 Device tree initial references information of RT-DMAC0 (R-Car V3U)

rt\_dmac1: dma-controller@ffd61000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-rt-dmac";

reg = <0 0xffd61000 0 0x1000>,

<0 0xffc20000 0 0xf104>;

interrupts = <GIC\_SPI 9 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 80 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 81 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 82 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 83 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 84 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 85 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 86 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 87 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 88 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 89 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 90 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 91 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 92 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 93 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 94 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 95 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 631>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 631>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-14 Device tree initial references information of RT-DMAC1 (R-Car V3U)

rt\_dmac2: dma-controller@ffd62000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-rt-dmac";

reg = <0 0xffd62000 0 0x1000>,

<0 0xffd70000 0 0xf104>;

interrupts = <GIC\_SPI 10 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 96 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 97 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 98 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 99 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 100 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 101 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 102 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 103 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 104 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 105 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 106 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 107 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 108 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 109 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 110 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 111 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 700>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 700>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-15 Device tree initial references information of RT-DMAC2 (R-Car V3U)

rt\_dmac3: dma-controller@ffd63000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-rt-dmac";

reg = <0 0xffd63000 0 0x1000>,

<0 0xffd80000 0 0xf104>;

interrupts = <GIC\_SPI 11 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 112 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 113 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 114 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 115 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 116 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 117 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 118 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 119 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 120 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 121 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 122 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 123 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 124 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 125 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 126 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 127 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 701>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 701>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-16 Device tree initial references information of RT-DMAC3 (R-Car V3U)

Basic configuration information of R-Car V3H SYS-DMAC is defined at device tree file (r8a77980.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

|  |
| --- |
| dmac1: dma-controller@e7300000 {  compatible = "renesas,dmac-r8a77980",  "renesas,rcar-dmac";  reg = <0 0xe7300000 0 0x10000>;  interrupts = <GIC\_SPI 220 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 216 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 217 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 218 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 219 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 308 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 309 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 310 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 311 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 353 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 354 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 355 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 356 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 357 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 358 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 359 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 360 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 218>;  clock-names = "fck";  power-domains = <&sysc R8A77980\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-17 Device tree initial references information of SYS-DMAC1 (R-Car V3H)

|  |
| --- |
| dmac2: dma-controller@e7310000 {  compatible = "renesas,dmac-r8a77980",  "renesas,rcar-dmac";  reg = <0 0xe7310000 0 0x10000>;  interrupts = < GIC\_SPI 307 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 312 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 313 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 314 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 315 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 316 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 317 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 318 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 319 IRQ\_TYPE\_LEVEL\_HIGH>;  GIC\_SPI 361 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 362 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 363 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 364 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 365 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 366 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 367 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 368 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 217>;  clock-names = "fck";  power-domains = <&sysc R8A77980\_PD\_ALWAYS\_ON>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-18 Device tree initial references information of SYS-DMAC2 (R-Car V3H)

Basic configuration information of R-Car V3H RT-DMAC should be defined at device tree file (r8a77980.dtsi). It exists at arch/arm64/boot/dts/renesas directory. This module uses these described contents in device tree file.

|  |
| --- |
| rt\_dmac0: dma-controller@ffc10000 {  compatible = "renesas,dmac-r8a77980",  "renesas,rcar-dmac";  reg = <0 0xffc10000 0 0x10000>;  interrupts = <GIC\_SPI 199 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 200 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 201 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 202 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 203 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 204 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 205 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 206 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 207 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 208 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 209 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 210 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 211 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 212 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 213 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 214 IRQ\_TYPE\_LEVEL\_HIGH  GIC\_SPI 215 IRQ\_TYPE\_LEVEL\_HIGH>;  interrupt-names = "error",  "ch0", "ch1", "ch2", "ch3",  "ch4", "ch5", "ch6", "ch7",  "ch8", "ch9", "ch10", "ch11",  "ch12", "ch13", "ch14", "ch15";  clocks = <&cpg CPG\_MOD 21>;  clock-names = "fck";  power-domains = <&sysc R8A77980\_PD\_ALWAYS\_ON>;  resets = <&cpg 21>;  #dma-cells = <1>;  dma-channels = <16>;  }; |

Figure 5-19 Device tree initial references information of RT-DMAC0 (R-Car V3H)

rt\_dmac1: dma-controller@ffc20000 {

compatible = "renesas,dmac-r8a77980",

"renesas,rcar-dmac";

reg = <0 0xffc20000 0 0x10000>;

interrupts = <GIC\_SPI 320 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 321 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 322 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 323 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 324 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 325 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 326 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 327 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 328 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 329 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 330 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 331 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 332 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 333 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 334 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 335 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 336 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 16>;

clock-names = "fck";

power-domains = <&sysc R8A77980\_PD\_ALWAYS\_ON>;

resets = <&cpg 16>;

#dma-cells = <1>;

dma-channels = <16>;

};

Figure 5-20 Device tree initial references information of RT-DMAC1 (R-Car V3H)

To enable fixed address mode support, “fixed-source” (for fixed source address) or “fixed-dest” (for fixed destination address) optional properties need to be added as following in corresponding DMAC device tree node.

fixed-source;

fixed-dest;

To enable slow speed mode (only available on R-Car V3U), “rate-read” (for read rate control) or “rate-write” (for write rate control) optional properties need to be added as following in corresponding DMAC device tree node. The valid value is in [0x03, 0xff] and inverse ratio to the speed which means the greater value is, the less speed is.

For example:

rate-read = <0x11>;

rate-write = <0xff>;

dmac1: dma-controller@e7350000 {

compatible = "renesas,dmac-r8a779a0”

"renesas,rcar-v3u-dmac";

reg = <0 0xe7350000 0 0x1000>,

<0 0xe7300000 0 0xf104>;

interrupts = <GIC\_SPI 6 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 32 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 33 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 34 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 35 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 36 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 37 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 38 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 39 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 40 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 41 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 42 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 43 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 44 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 45 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 46 IRQ\_TYPE\_LEVEL\_HIGH

GIC\_SPI 47 IRQ\_TYPE\_LEVEL\_HIGH>;

interrupt-names = "error",

"ch0", "ch1", "ch2", "ch3",

"ch4", "ch5", "ch6", "ch7",

"ch8", "ch9", "ch10", "ch11",

"ch12", "ch13", "ch14", "ch15";

clocks = <&cpg CPG\_MOD 709>;

clock-names = "fck";

power-domains = <&sysc R8A779A0\_PD\_ALWAYS\_ON>;

resets = <&cpg 709>;

#dma-cells = <1>;

dma-channels = <16>;

**fixed-source;**

**fixed-dest;**

**rate-read = <0x11>;**

**rate-write = <0xff>;**

};

};

Figure 5-21 Enable fixed address mode, slow speed mode references information of SYS-DMAC1 (R-Car V3U)

## Option Setting

### Module Parameters

There are no module parameters.

### Kernel Parameters

There are no module parameters.