1. Overview

1.1 Overview

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

1.2 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

1.3 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.

1.4 Reference

1.4.1 Standards

There is no reference document on standards.

1.4.2 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

1.5 Restrictions

There is no restriction in this module.

2. 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

3. Operating Environment

3.1 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

3.2 Module Configuration

The following figures show the configuration of this module.

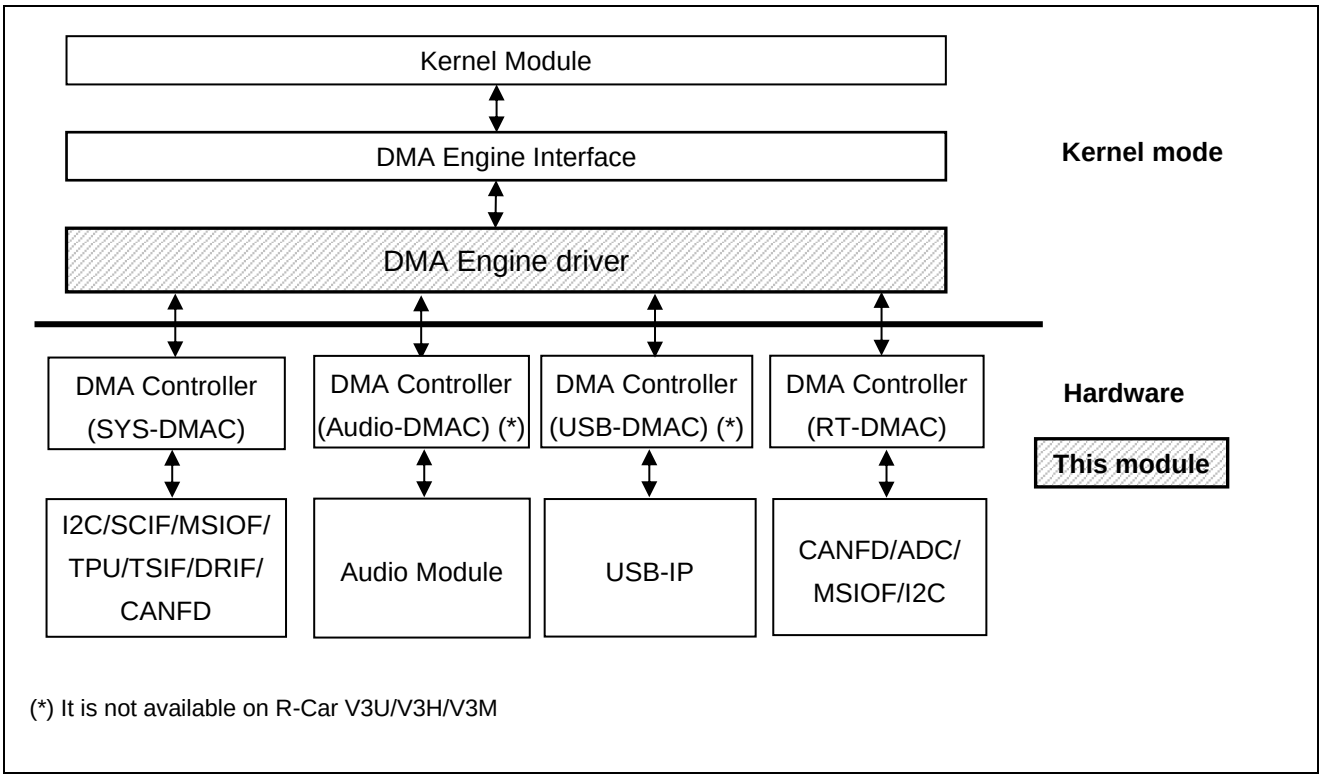

Figure 3-1 DMA Engine Driver configuration (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

3.3 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.

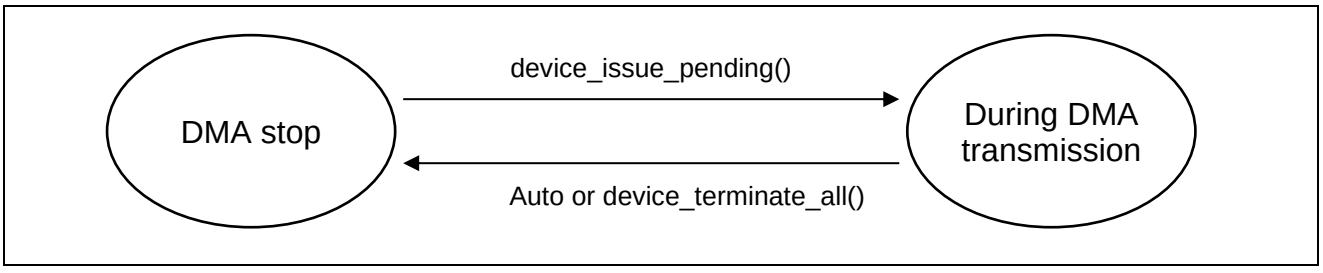

Figure 3-2 DMA Engine Driver State Transition Diagram (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

4. 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.

4.1 Device Node

There are no device nodes in this module.

4.2 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.

4.2.1 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] -

4.2.2 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] -

4.2.3 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.

4.2.4 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] -

4.2.5 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] -

4.2.6 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] -

4.2.7 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] -

4.2.8 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] -

4.2.9 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] -

4.2.10 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] -

4.2.11 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] -

4.2.12 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]

4.2.13 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] -

4.2.14 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] -

4.2.15 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.

4.2.16 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.

4.2.17 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.

4.2.18 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.

4.2.19 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] -

4.2.20 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.

4.3 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.

4.3.1 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] -

4.3.2 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] -

4.3.3 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] -

4.3.4 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] -

4.3.5 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] -

4.3.6 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] -

4.3.7 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] -

4.3.8 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] -

4.3.9 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.

4.3.10 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] -

4.4 Structure

4.4.1 This module structures

This section shows the structure that this module is defined.

4.4.1.1 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)

4.4.1.2 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)

4.4.1.3 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)

4.4.1.4 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)

4.4.1.5 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)

4.4.1.6 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)

4.4.1.7 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)

4.4.1.8 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)

4.4.1.9 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)

4.4.1.10 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)

4.4.1.11 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)

4.4.1.12 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)

4.4.2 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.

4.4.2.1 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)

4.4.2.2 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)

4.4.2.3 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)

4.4.2.4 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)

4.4.2.5 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)

4.4.2.6 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)

4.4.2.7 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)

4.4.2.8 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)

4.4.2.9 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)

4.4.2.10 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)

4.5 Global Variables and Constants

4.5.1 Global Variables

There are no global variables for this module.

4.5.2 Global Constants

This section shows global constants that this module is used.

4.5.2.1 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

4.5.2.2 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.

4.5.2.3 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.

4.6 Example of use

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

4.6.1 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)

4.6.2 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)

4.6.3 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)

4.6.4 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)

4.6.5 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)

4.6.6 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)

4.6.7 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)

4.6.8 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)

5. Integration

5.1 Directory Configuration

The directory configuration is shown below.

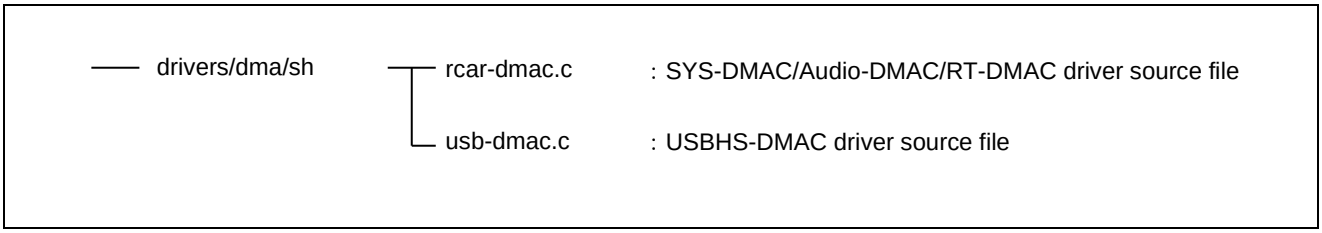

Figure 5-1 Directory configuration (R-Car H3/M3/M3N/E3/D3/V3U/V3H)

5.2 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)

5.3 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)

5.4 Option Setting

5.4.1 Module Parameters

There are no module parameters.

5.4.2 Kernel Parameters

There are no module parameters.