User’s Manual

CONFIDENTIAL

IMP Framework

User’s Manual: Software

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

Renesas Electronics
www.renesas.com

Rev.0.20E Jul. 2021

Trademark

All trademarks and registered trademarks are the property of their respective owners.

- Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- Other product names, company names, and brands described in this manual are owned by the respective companies.
- Trademarks and trademark symbols (® or ™) are omitted in the text of this manual.

How to use this manual

1. Objective

This manual is a manual for users to understand the user interface specification of this software and targets for users to design application systems used by this software. Please refer to the manuals related to this software.

2. Usage for this software

In case of using this software, the customer must make an agreement for software licensing with our company.

3. Abbreviation

- *k* with number means 1000.
- *K* means 1024.
- *NULL* means a pointer to 0 address.
- A *mod* B means a remainder with A divided by B.
- About mathematical formulas in functional description for API, *i* means index of sequences and c means channel number.

Table of Contents

1. Overview

1.1 Feature

1.1.1 IMP related software

Figure 1-1 describes the entire structure of software to control IMP-Xn. User application uses libraries which are consist of "Atomic Library", "IMP Framework" and "IMP Driver".

First, user application calls Atomic Library to generate command lists which are executed on IMP-Xn. Second, user application calls IMP Framework with the command lists generated by Atomic Library.

IMP Framework schedules execution order of Command lists or Runtime Test, and calls IMP Driver or IMP RTT Driver.

If select the command list control, IMP Driver start executing IMP-Xn, retrieve execution result from IMP-Xn and return it to IMP Framework.

If select the runtime test control, IMP RTT Driver start executing Field BIST Module, retrieve execution result from Field BIST Module and return it to IMP Framework.

Finally, IMP Framework returns the result to user application.

This document explains about the Software Architecture Design of IMP Framework in red box.

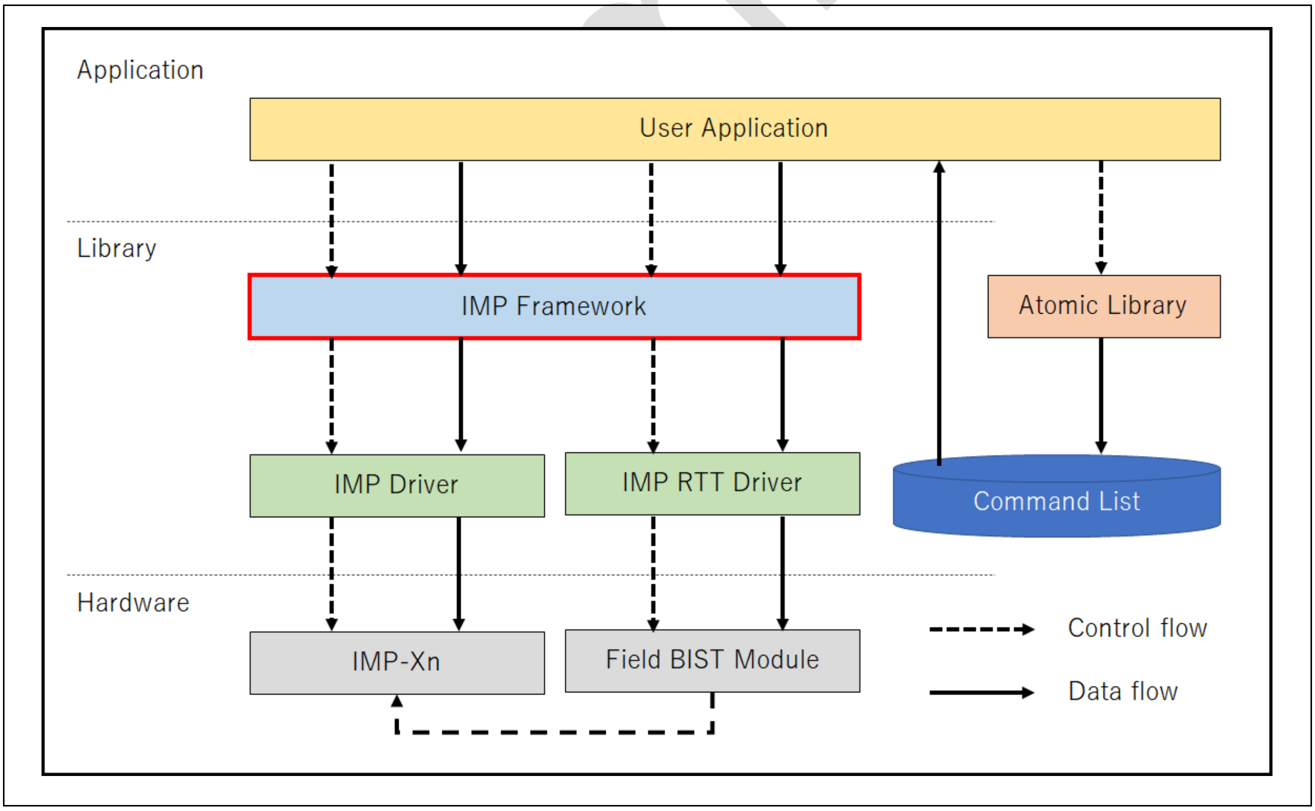


Figure 1-1 Entire structure of software to control IMP-Xn

1.1.2 IMP Framework

This software is one of the Cognitive (recognition) software which controls IMP-Xn. This software provides API of IMP Framework. The Block Diagram of IMP Framework is shown in ***Figure 1-2***.

Component boxed in red is the scope of this document.

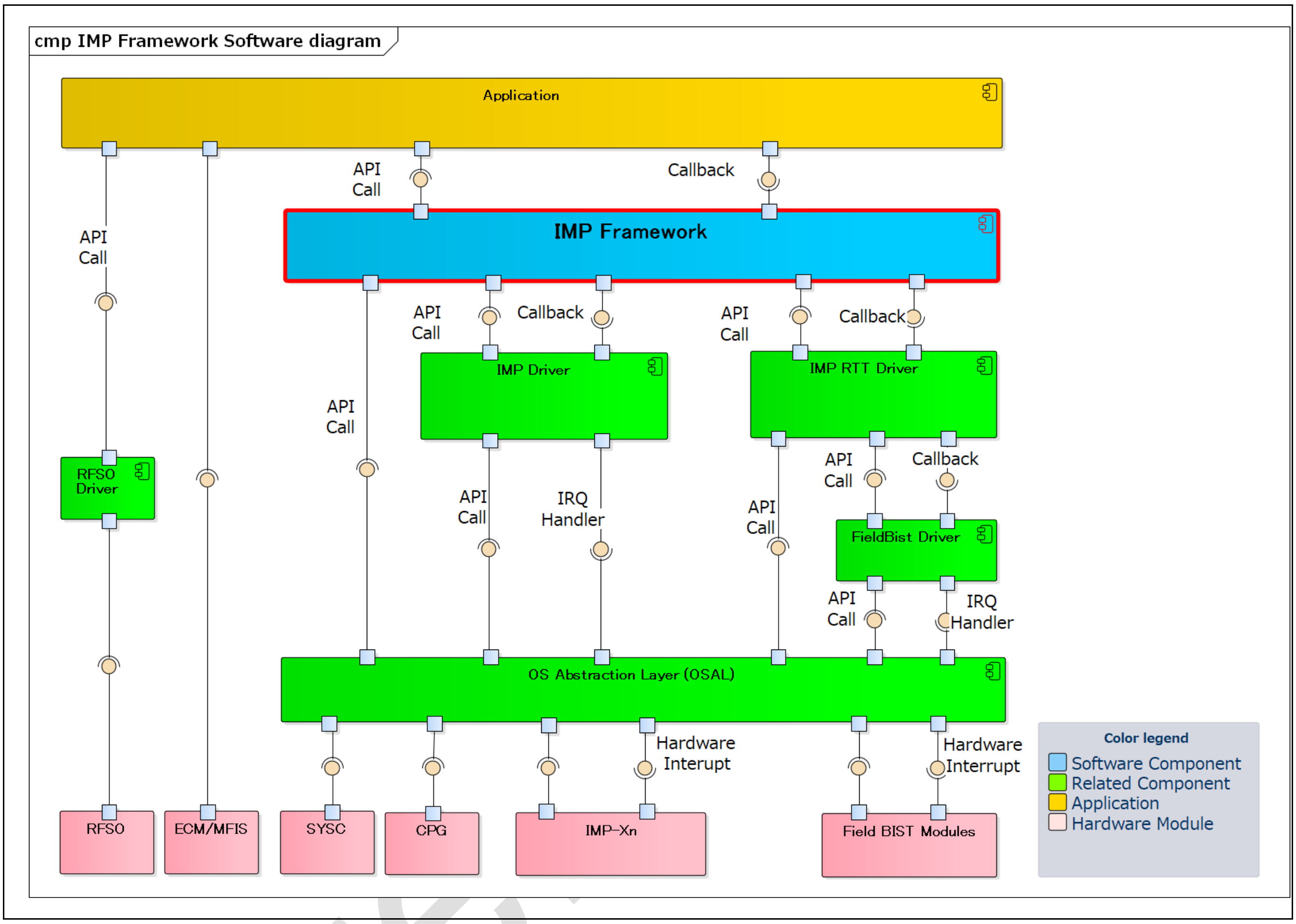


Figure 1-2 Block Diagram

The following ***Table 1-1*** gives an outline of each component. Details of each component refer to each document.

Table 1-1 Component List

Component	Description
User Application	User Application to control IMP-Xn function API. Manage generation of CL data, and enter CL data into IMP Framework.
IMP Framework	Provides the API of IMP Framework. This software provides scheduler function to manage several CL for IMP-Xn together.
IMP Driver	For details about IMP Driver specification.
OS Abstraction Layer	Refer to the ***Operating System Abstraction Layer(OSAL) API Application Note*** for detail.
IMP RTT Driver	For details about IMP RTT Driver specification (Not implement).
Field BIST Driver, RFSO Driver	Out of scope of this document
IMP-Xn, CPG, SYSC, Field BIST modules, RFSO, ECM/MFIS	Out of scope of this document

1.2 References

Table 1-2 References File List

No.	Title	Version
[1]	IMP Framework Product Information for each R-car V3 series	V3M: Rev.0.18E V3H: Rev.0.18E V3Hv2: Rev.0.18E V3U: Rev.0.18E
[2]	Atomic Library User’s Manual	T.B.D.
[3]	Operating System Abstraction Layer(OSAL) API Application Note	Ver 0.40E
[4]	Operating System Abstraction Layer API(OSAL API)	Ver.0.60E

1.3 List of Terms

Table 1-3 Terminology and Abbreviations List

Abbreviation	Description
CL	Command List
HW	Hardware
IMP	Image Processing Unit
IMP-Xn	Image Processing Unit extended “n”. “n” depends on the SoC. Refer to [1]
INT	Interrupt generation command
SLP	An instruction of CL. It is a part of Inter-Module Synchronization feature of IMP-Xn.
SW	Software
WUP	An instruction of CL. It is a part of Inter-Module Synchronization feature of IMP-Xn.
Runtime Test	The Runtime Test is a safety mechanism to detect permanent faults during the SoC is used.
Safety Mechanism	The security mechanism consists of an IP security mechanism, an on-chip security mechanism, and a system-level security mechanism.
DTA	Debug Trace Agent.
OSAL	Operating System Abstraction Layer

2. Basic Specification

2.1 Summary Specification

The IMP Framework provides is following functions for the Execution of command list control on IMP-Xn hardware.

- Public functions
  - ➢ Initialize and exit the IMP Framework
    - ✧ Provide the ***R_IMPFW_Init*** function that initializes of IMP Framework. This function initializes is OSAL resources and control tables used inside the IMP Framework, and also initializes the IMP Driver that controls of IMP-Xn hardware.
    - ✧ Provide the ***R_IMPFW_Quit*** function that terminates of IMP Framework. This function also releases is OSAL resources used inside of IMP Framework and terminates the IMP Driver.
  - ➢ Attribute data setting function when executing the command list
    - ✧ Following functions are provided to set attribute data, when execute the command list.
      - ●***R_IMPFW_AttrInit***
        - ➢ Initialize with this function before setting the attribute data.
      - ●***R_IMPFW_AttrSetCl***
        - ➢ Set the CL physical address, CL size, and priority for command list.
      - ●***R_IMPFW_AttrSetPair***
        - ➢ Set the Pair core list for executes at command list.
      - ●***R_IMPFW_AttrSetCoremap***
        - ➢ Set the Core map for synchronization used of 'WUP / SLP' and 'DPR' instructions in Command list.
      - ●***R_IMPFW_AttrSetInterrupt(Not implement)***
        - ➢ Set when using the Group interrupt and interrupt mask functions in IMP-Xn hardware.
  - ➢ Execution function for Command list
    - ✧ Provide is ***R_IMPFW_Execute*** function to execute the command list. This function is execution of the command list in an asynchronous control, and if ***R_IMPFW_Execute*** function is called during the execution of command list, it is stored and executed sequentially using the queue system inside IMP Framework.
    - ✧ Since command list execution is controlled asynchronously, notifications such as command list completion (TRAP instruction decode), command list stop (INT instruction decode), and command list execution error are specified by ***R_IMPFW_Execute*** function callback function. It is done with ***p_impfw_cbfunc_t***.
    - ✧ It also provides ***R_IMPFW_Resume(Not implement)*** function to resume execution of the Command list when the command list is stopped at INT instruction decode.

- Support for IMP-Xn interrupt function
  - ➢ Multi-channel interrupt function
    - ✧ Possible to select the interrupt number (Interrupt ID) to be used for interrupting from IMP-Xn hardware.
    - ✧ Interrupt number is selected by instance number (***e_impfw_instance_t***) in the p_initdata argument of ***R_IMPFW_Init*** function.
    - ✧ Refer to '***Product Information***' because the interrupt numbers that can be specified differ depending on the product.

- Group interrupt function
  - ✧ Possible to use the interrupt group function of IMP-Xn hardware.
  - ✧ Group Interrupt is set by calling to ***R_IMPFW_AttrSetInterrupt(Not implement)*** function.
  - ✧ Relation the Group number and IMP-Xn core is differs of depending on product, so refer to '***Product Information***'.
- Interrupt mask setting function
  - ✧ Possible to use the interrupt mask function of IMP-Xn hardware.
  - ✧ Interrupt factors of other than TARP, INT, and IER, Can be set by calling ***R_IMPFW_AttrSetInterrupt(Not implement)*** function.
  - ✧ Relation the Interrupt factors and IMP-Xn core is differs of depending on product, so refer to '***Product Information***'.

● Support for CL synchronization function on IMP-Xn core

- Pair core settings
  - ✧ Possible to Setting the Pair core list for executes at command list.
  - ✧ For details on Pair core settings, refer to '***2.Pair Function***' in ***Appendix***.
- Core map settings
  - ✧ Possible to Setting the Core map for synchronization used of 'WUP/SLP' and 'DPR' instructions in Command list.
  - ✧ For details on Pair core settings, refer to '***4.Setting CoreMap***' in ***Appendix***.
  - ✧ Core map to set is differs of depending on product, so refer to '***Product Information***'.

● Power policy setting function

- Provide of ***R_IMPFW_SetPmPolicy(Not implement)*** function that sets the power policy for IMP-Xn core.
- This function allows to select a power policy (High Performance / Clock Gate / Power Gate), based on the function of 'OSAL Power Manager'.
- IMP Framework is starts and stops the IMP-Xn core, when the command list is executed and terminated, so control by the power policy is applied at this time.

● Hardware error check function

- IMP Framework is provides Hardware error checking function by Runtime Test and Safety Mechanism.
- However, it is not implemented at this time.

● Software version acquisition function

- Provide is ***R_IMPFW_GetVersion*** function to acquisition software version of IMP Framework.

2.2 Reserved word

The IMP Framework uses the following prefixes for avoiding confusion from other software. Prefixes is described in ***Table 2-1***.

Table 2-1 Prefixes

prefix	Description
R_IMPFW_***	External functions
impfw_***	Internal functions
Impfw_***_t	Basic type
st_impfw_***	Structure type
e_impfw_***	Enumeration type
IMPFW_***	Enumerators
IMPFW_***	Define
s_impfw_***	Global Variables (Read Only)
p_impfw_***	Function pointer type

2.3 Cautions of Software

Table 2-2: Cautions Item List

No.	Items	contents
(1)	Timeout function	This software does not detect timeout. Therefore, users need to implement timeout processing at the upper layer (such as the application side) that uses this software.

3. Function Description

3.1 Finite-State machine

This software has five states for each core. The following ***Figure 3-1*** shows state machine.

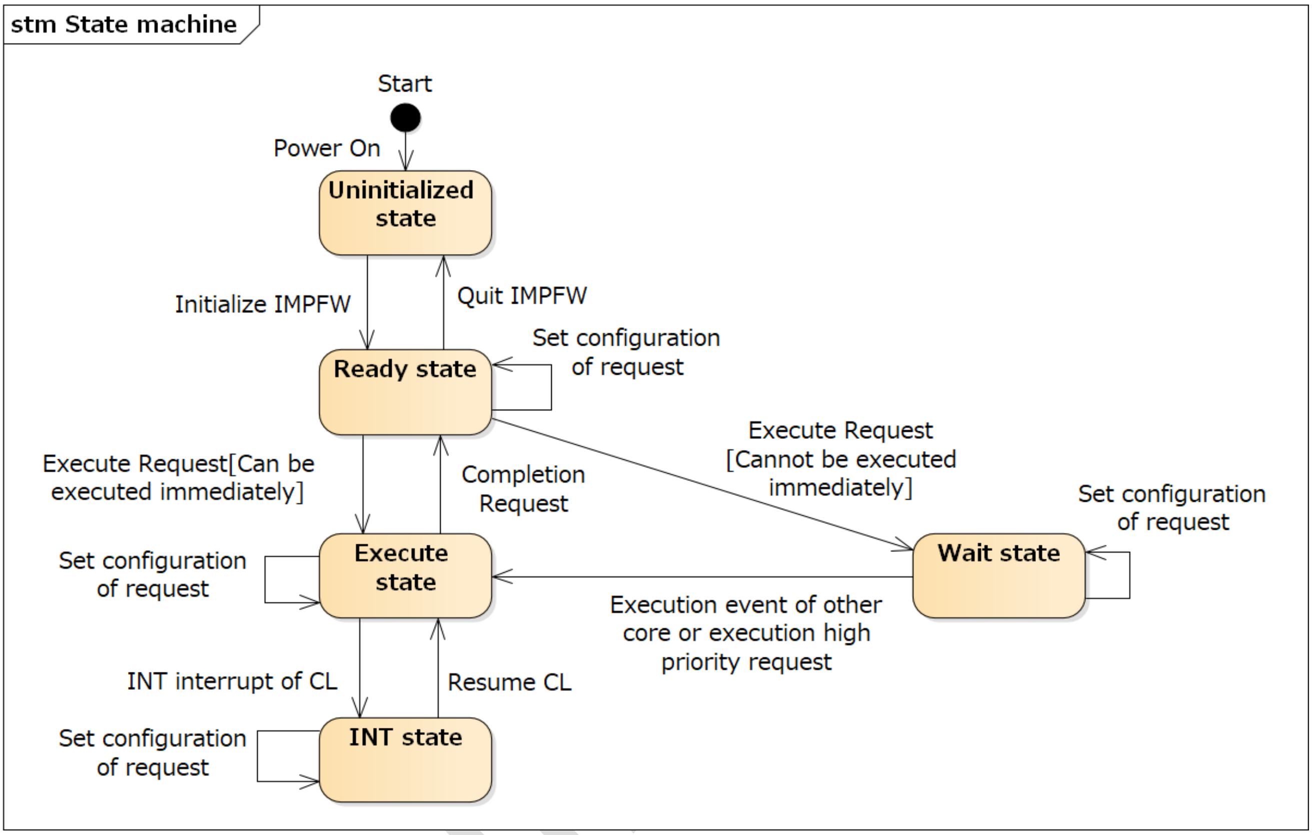

stm State machine
Start
Power On
Uninitialized state
Initialize IMPFW
Quit IMPFW
Ready state
Set configuration of request
Execute Request[Can be executed immediately]
Completion Request
Execute Request [Cannot be executed immediately]
Set configuration of request
Set configuration of request
Execute state
Wait state
Execution event of other core or execution high priority request
INT interrupt of CL
Resume CL
Set configuration of request
INT state

Figure 3-1 State machine

3.1.1 Uninitialized State

This state is the default software state immediately after booting the system. By calling ***R_IMPFW_Init***

, the state of specified core will be changed to ***Ready State***

.

3.1.2 Ready State

This state is the ready state and is possible to start processing. It is a state indicating that the core is not execute processing. By calling ***R_IMPFW_Execute***

, the state of core will be changed.

3.1.3 Wait State

This is the state that is waiting the pair core. When the either of following conditions is satisfied the state of core will be changed.

- When the pair CL or Runtime Test request is enqueued to the CL queue.
- When a "not pair CL" or "not pair Runtime Test" request with a higher priority than the CL used in pairing is enqueued to the CL queue.

3.1.4 Execute State

This state is an execute processing and shows that it is execution of the core. When the core processing is completed, it calls ***p_impfw_cbfunc_t***

and the status returns to ***Ready State***

. The User Application is able to handle IMP-Xn even in this state.

3.1.5 INT State

This state is INT interrupt state. User application is able to add CL or Runtime Test request to queue by ***R_IMPFW_Execute***

call in this state. By calling ***R_IMPFW_Resume***(Not implement)

, the state of core will be changed.

3.2 Function Flow

The user initializes the FW with R_IMPFW_Init and initializes the Attribute information with ***R_IMPFW_AttrInit***. When executing multiple CL requests, execute ***R_IMPFW_AttrInit*** for the number of requests. Then, using the Attribute handle initialized with ***R_IMPFW_AttrInit***, call ***R_IMPFW_AttrSetCl***(other R_IMPFW_AttrSetXXX is optional) and ***R_IMPFW_Execute*** to execute CL.

Attribute information can be changed with R_IMPFW_AttrSetXXX after receiving the CL execution completion callback. Also, if you want to change the association between Attribute handle and core, execute it from ***R_IMPFW_AttrInit*** after ***R_IMPFW_Quit***. Please refer to ***Table 3-1*** No.3 for Attribute handle.

3.2.1 Basic Execution

The typical function-flow is shown in ***Figure 3-2***. In addition, "Wait processing" can be executed by any method (ex. Message communication).For more function details, please refer to ***Function***.

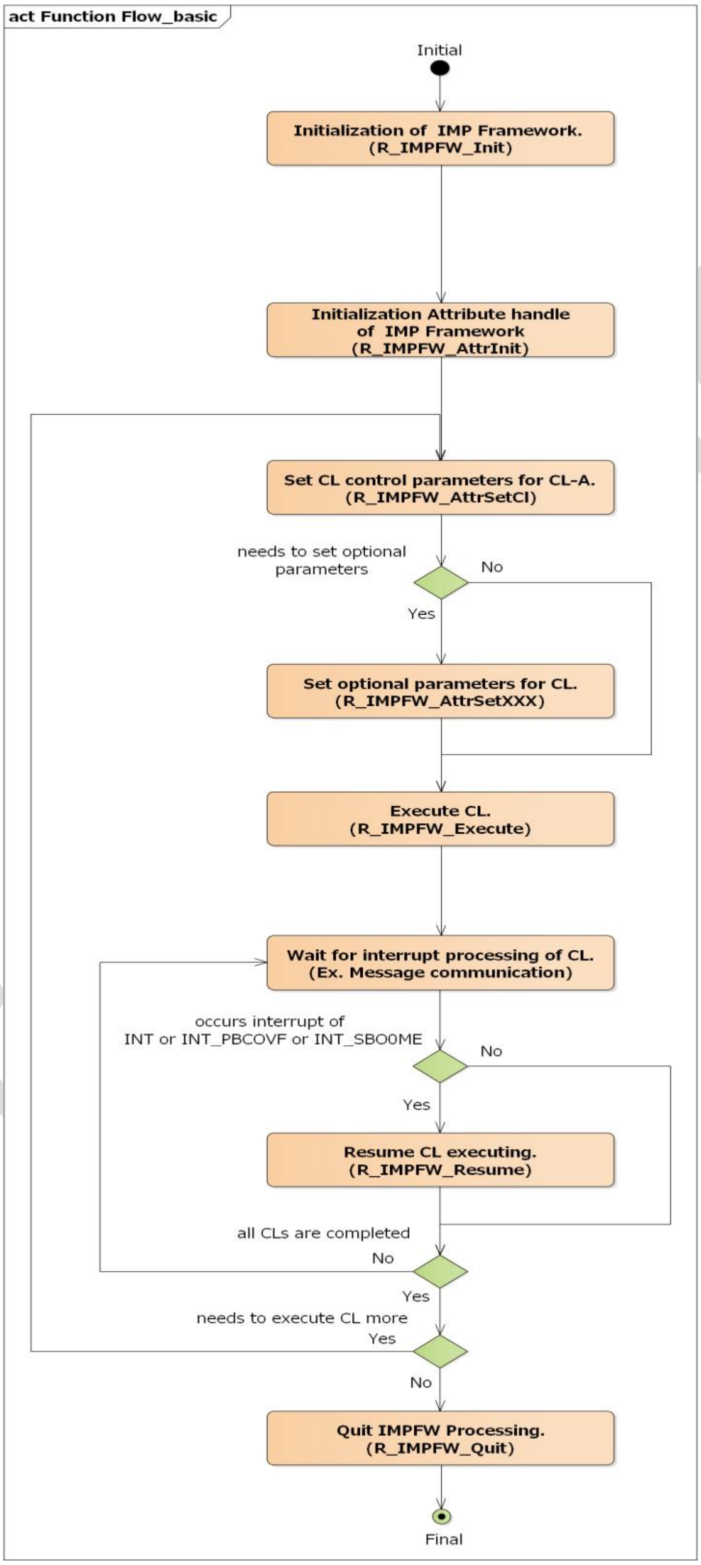
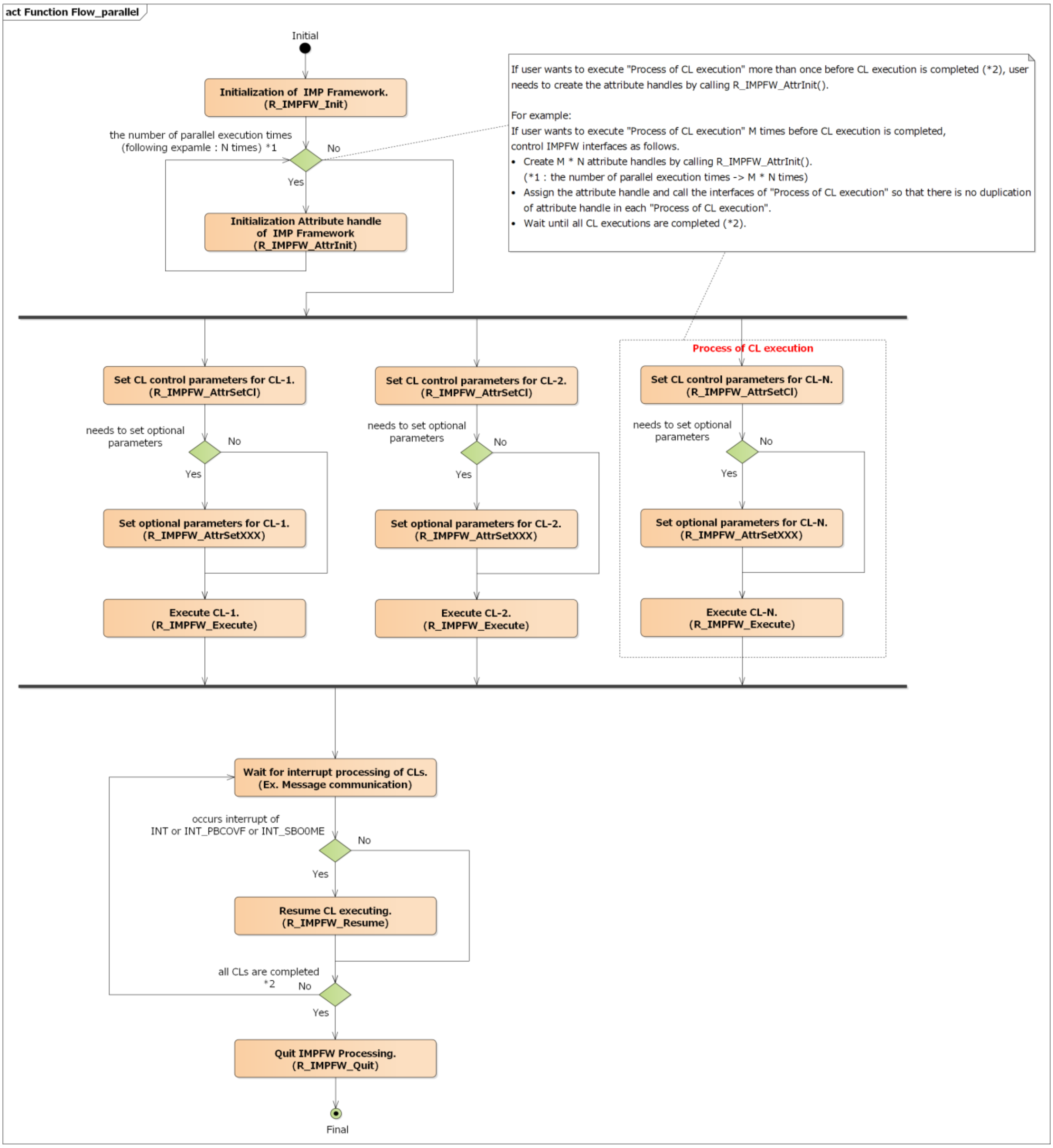


Figure 3-2 Function Flow diagram

3.2.2 Parallel Execution

The flow of parallel execution is shown in ***Figure 3-3***. In addition, "Wait processing" can be executed by any method (ex. Message communication).


act Function Flow_parallel
Initial
Initialization of IMP Framework.
(R_IMPFW_Init)
the number of parallel execution times
(following expamle : N times) *1
No
Yes
Initialization Attribute handle
of IMP Framework
(R_IMPFW_AttrInit)
Process of CL execution
Set CL control parameters for CL-1.
(R_IMPFW_AttrSetCl)
needs to set optional parameters
No
Yes
Set optional parameters for CL-1.
(R_IMPFW_AttrSetXXX)
Execute CL-1.
(R_IMPFW_Execute)
Set CL control parameters for CL-2.
(R_IMPFW_AttrSetCl)
needs to set optional parameters
No
Yes
Set optional parameters for CL-2.
(R_IMPFW_AttrSetXXX)
Execute CL-2.
(R_IMPFW_Execute)
Set CL control parameters for CL-N.
(R_IMPFW_AttrSetCl)
needs to set optional parameters
No
Yes
Set optional parameters for CL-N.
(R_IMPFW_AttrSetXXX)
Execute CL-N.
(R_IMPFW_Execute)
Wait for interrupt processing of CLs.
(Ex. Message communication)
occurs interrupt of
INT or INT_PBCOVF or INT_SBOOME
No
Yes
Resume CL executing.
(R_IMPFW_Resume)
all CLs are completed
*2
No
Yes
Quit IMPFW Processing.
(R_IMPFW_Quit)
Final

If user wants to execute "Process of CL execution" more than once before CL execution is completed (*2), user needs to create the attribute handles by calling R_IMPFW_AttrInit().

For example:
If user wants to execute "Process of CL execution" M times before CL execution is completed, control IMPFW interfaces as follows.

- Create M * N attribute handles by calling R_IMPFW_AttrInit().
  (*1 : the number of parallel execution times -> M * N times)
- Assign the attribute handle and call the interfaces of "Process of CL execution" so that there is no duplication of attribute handle in each "Process of CL execution".
- Wait until all CL executions are completed (*2).

Figure 3-3 Function Flow diagram of parallel execute

3.3 Error Processing

If some error is detected in IMP Framework, this software notifies the user of an error by the result value of the function. This section shows about each error.

3.3.1 PARAMETER ERROR

This error occurs by specifying an invalid parameter.

If this error occurs, IMP Framework will return ***IMPFW_EC_NG_PARAM***.

In this case, the user can continue by put the correct value to the argument.

Please refer to ***Function*** for detail of each parameter.

3.3.2 SYSTEM ERROR

This error occurs by hardware error in IMP-Xn, OSAL function error or Initialize sequence error.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_SYSTEMERROR***.

When this error occurred by not performing the initialization sequence, modify the function order to call the function from the correct state machine.

Other case, execute system reset.

3.3.3 SEQUENCE (STATE) ERROR

This error occurs by illegal state transition.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_SEQSTATE***.

In this case, modify the function order to call the function from the correct state machine.

Please refer to ***Finite-State machine*** and ***Function*** for the correct combination of function call and state machine.

3.3.4 IMP DRIVER ERROR

This error occurs when the IMP Driver returned error.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_DRVERROR***.

In this case, it is necessary to check the data related to the IMP Driver.

If it occurs in R_IMPFW_Init, check the setting of ***st_impfw_drv_resource_t***.

If it occurs in any other API, take the same action as ***SYSTEM ERROR***.

3.3.5 IMP RTT DRIVER ERROR (Not implement)

This error occurs when the IMP RTT Driver returned error.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_RTTERROR***

In this case, it is necessary to check the data related to the IMP RTT Driver.

3.3.6 ATTRIBUTE ERROR

This error occurs when the combination of attributes is abnormal.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_ATTRERROR***.

In this case, the user can continue by set the attribute settings with the correct combination.

Please refer to description of ***R_IMPFW_AttrInit*** about attribute.

3.3.7 NOT SUPPORT ERROR

This error occurs when the user uses a feature that the hardware does not support.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_NOTSUPPORT***.

3.3.8 RESOURCE FULL ERROR

This error occurs when the message queue or ***Management memory area*** is exhausted.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_RESOURCEFULL***.

In this case, modify “max_queue_num” or “max_msg_num” of ***st_impfw_fw_resource_t*** at calling ***R_IMPFW_Init***

.

3.3.9 TIMEOUT ERROR

This error occurs by mutex timeout or message timeout.

If this error occurs, the IMP Framework will return ***IMPFW_EC_NG_TIMEOUT***.

In this case, modify the timeout values at calling ***R_IMPFW_Init***

.

Please refer to ***Function*** for detail of each parameter.

3.4 Usage and Restriction

3.4.1 User side prepare data

Before using IMP Framework, show the preparation of the User Application below. When using the IMP Framework, User Application need to prepare the data in the following ***Table 3-1*** on the user side, and set it in the argument of ***R_IMPFW_Init***

function.

Table 3-1 User side prepare data List

No.	Input Parameter	Description	Notes
1	Control Handle	IMP Framework's control handle information defined in ***impfw_ctrl_handle_t***.	(1) The User Application sets this handle in the argument of ***R_IMPFW_Init*** (2) and initializes the IMP Framework. After this, The User Application must set the same handle for each API argument. (3) The User Application must keep this handle unchanged until it runs ***R_IMPFW_Quit*** (4).
2	Management memory area (work memory area)	This is the data area that manages the IMP framework processing, and it must be prepared separately for each type of management area of the IMP framework. The attribute management area and the queue management area depend on the maximum number of queues that you reserve.	(1) The user side should set the start address pointer of this data as an argument of the ***R_IMPFW_Init*** (2) function. (3) The User Application can prepare only one this data in an instance. If IMP Framework is operated using multiple data, operation can not be guaranteed in an instance. (4) This data must be initialized with zero. (5) The address of the management memory area is specifying the memory address secured by 8-byte alignment. If it is not 8-byte alignment, return error. (6) The size of the required management memory area is as follows. ・ IMPFW_WORKAREA_TYPE_MAIN : 13KB ・ IMPFW_WORKAREA_TYPE_ATTR : 568B * max_queue_num of ***st_impfw_fw_resource_t*** ・ ・ IMPFW_WORKAREA_TYPE_QUE : 248B * max_queue_num of ***st_impfw_fw_resource_t*** ・
3	Attribute Handle	IMP Framework's attribute handle	The user application manages handles by

		information defined in ***impfw_attr_handle_t***. The User Application sets this handle in the argument of ***R_IMPFW_Execute*** and execute the request to IMP Framework. This handle is the handle associated with the request to IMP Framework. The user application must keep this handle as many as the number of requests.	following the steps below. Refer to ***Figure 3-2*** about flow. (1) The user application sets the number of requests to the IMP Framework (the maximum number that can be requested at the same time by specified all core) in the argument “max_queue_num” of ***R_IMPFW_Init*** (2). Ex. If max_queue_num is set to 50: The 50 queues are shared by all cores specified in p_initdata. (3) The user application calls ***R_IMPFW_AttrInit*** (4) for the number of requests to the IMP Framework and acquires the ***Attribute Handle***. In addition, it is necessary to keep the acquired handle unchanged until ***R_IMPFW_Quit*** (5) is executed. Ex. If max_queue_num is set to 50： If you want to use 10 queues of 50 queues in IMPCore0, you need to run ***R_IMPFW_AttrInit*** 10 times. The remaining 40 queues can be assigned to the other cores specified in ***R_IMPFW_Init*** .. (6) Set the ***Attribute Handle*** and request parameters in the argument of R_IMPFW_AttrSet*** (***R_IMPFW_AttrSetCl*** (7) etc..). Here, the ***Attribute Handle*** and the request are linked. (8) Set the ***Attribute Handle*** in the argument of ***R_IMPFW_Execute*** (9) and execute the request. The request parameter associated with this ***Attribute Handle*** cannot be changed until the completion callback is returned.

			(10) After the completion callback is notified, the ***Attribute Handle*** specified in ***R_IMPFW_Execute*** (11) in step 4 can be set according to step 3 again.

These data shown above is necessary for internal control of Framework. If these data are rewritten during Framework operation, operation can not be guaranteed.

3.4.2 Use of OSAL Resources

When using IMP Framework, it is necessary to register required ‘HW Resources’ and ‘OS Resources’ in OSAL Configurator.

The following table shows OSAL resources used by the IMP Framework.

Table 3-2 List of values for OSAL Thread resource ID

Thread ID	task name[chars]	stack size[Byte]
task_id[] of ***st_impfw_fw_resource_t***	"IMPFW MAIN CONTROL TASK"	task_stacksize of ***st_impfw_fw_resource_t***

Table 3-3 List of values for OSAL Mutex resource ID

Mutex ID	Description
mutex_id[] of ***st_impfw_fw_resource_t***	OSAL Configure ID of Mutex object.

Table 3-4 List of values for OSAL Message queue resource ID

Message ID	number of messages	message size[Byte]
0	max_msg_num ***st_impfw_fw_resource_t***	272
1	max_msg_num ***st_impfw_fw_resource_t***	16

4. Data Type definition

4.1 Basic type

Table 4-1 Definition Values

Type name	Description
impfw_ctrl_handle_t	IMP Framework Control handle
impfw_attr_handle_t	IMP Framework Attribute handle

4.2 Definition Values

Table 4-2 Definition Values

Name	Value	Description
IMPFW_COREMAP_MAXID	(16U)	Max number of core map.
IMPFW_MSGTYPE_NUM	(2U)	Max number of message type
IMPFW_MUTEXTYPE_NUM	(3U)	Max number of mutex type
IMPFW_TASKTYPE_NUM	(1U)	Max number of task type
IMPFW_WORKAREA_TYPE_NUM	(3U)	Max number of work area type
IMPFW_VERSION_MAJOR	(2U)	Major version of IMP Framework
IMPFW_VERSION_MINOR	(2U)	Minor version of IMP Framework
IMPFW_VERSION_PATCH	(0U)	Patch version of IMP Framework

4.3 Enumerated type

This section shows the enumerated type used on this software.

User can use enumeration, must not use immediate value.

4.3.1 e_impfw_api_retcode_t

This is return code on this software. In this software this enumerator is used for return value in the external function. For the error level information, refer to

Error Processing.

```
typedef enum {
        IMPFW_EC_OK                               = 0,
        IMPFW_EC_NG_PARAM
        IMPFW_EC_NG_SYSTEMERROR
        IMPFW_EC_NG_SEQSTATE
        IMPFW_EC_NG_DRVERROR
        IMPFW_EC_NG_RTTERROR
        IMPFW_EC_NG_ATTRERROR
        IMPFW_EC_NG_NOTSUPPORT
        IMPFW_EC_NG_RESOURCEFULL
        IMPFW_EC_NG_TIMEOUT
} e_impfw_api_retcode_t
;
```

Table 4-3 Enumerator of ***e_impfw_api_retcode_t***

Value	Description
IMPFW_EC_OK	Processing is successful.
IMPFW_EC_NG_PARAM	Parameter error
IMPFW_EC_NG_SYSTEMERROR	System error(OSAL error).
IMPFW_EC_NG_SEQSTATE	API call does not follow state transition.
IMPFW_EC_NG_DRVERROR	IMP driver error.
IMPFW_EC_NG_RTTERROR	IMP RTT driver error.
IMPFW_EC_NG_ATTRERROR	Insufficient combinations and settings that are NG in control.
IMPFW_EC_NG_NOTSUPPORT	Unsupported features or cores specified.
IMPFW_EC_NG_RESOURCEFULL	The queue resource or the message resource is full.
IMPFW_EC_NG_TIMEOUT	Time out error.

4.3.2 e_impfw_callback_reason_t

The following table lists callback reason codes used by arguments of callback function.

The valid callback reasons depend on the SoC.

Refer to ***IMP Framework Product Information: Callback reason*** for details.

```
typedef enum {
        IMPFW_CB_STARTED                                 = 1,
        IMPFW_CB_TRAP                                    = 2,
        IMPFW_CB_INT                                     = 3,
        IMPFW_CB_INT_PBCOVF                              = 4,
        IMPFW_CB_ERROR_ILLEGAL                           = 5,
        IMPFW_CB_ERROR_INTERNAL                          = 6
        IMPFW_CB_USIER                                   = 7,
        IMPFW_CB_INT_SBO0ME                              = 8,
        IMPFW_CB_TRAP_SBO0ME                             = 9,
        IMPFW_CB_PBCOVF                                  = 10,
        IMPFW_CB_WUPCOV                                  = 11,
        IMPFW_CB_HPINT                                   = 12,
        IMPFW_CB_APIPINT                                 = 13,
        IMPFW_CB_USINT                                   = 14,
        IMPFW_CB_END                                     = 15,
        IMPFW_CB_MSCO                                    = 16,
        IMPFW_CB_UDIVSBRK                                = 17,
        IMPFW_CB_UDIPSBRK                                = 18,
        IMPFW_CB_DRVERR                                  = 19,
        IMPFW_CB_RTTERR                                  = 20,
        IMPFW_CB_RESOURCEFULL                            = 21
 } e_impfw_callback_reason_t
 ;
```

Table 4-4 Enumerator of ***e_impfw_callback_reason_t***

Value	Description
IMPFW_CB_STARTED	CL processing is started on hardware.
IMPFW_CB_TRAP	CL processing is completed normally.
IMPFW_CB_INT	CL process completes with INT command state.
IMPFW_CB_INT_PBCOVF	CL process completes with INT command state and Performance Busy Counter Overflow.
IMPFW_CB_ERROR_ILLEGAL	CL process completes with illegal command.
IMPFW_CB_ERROR_INTERNAL	CL processing is failed to start.
IMPFW_CB_USIER	CL process detects USIER error.
IMPFW_CB_INT_SBO0ME	CL process completes with INT command state and detect SBO0ME.
IMPFW_CB_TRAP_SBO0ME	CL processing is completed normally and detect SBO0ME.
IMPFW_CB_PBCOVF	CL process detects PBCOVF.
IMPFW_CB_WUPCOV	CL process detects WUPCOV.
IMPFW_CB_HPINT	CL process detects HPINT.
IMPFW_CB_APIPINT	CL process detects APIPINT.

IMPFW_CB_USINT	CL process detects USINT.
IMPFW_CB_END	CL process detects END.
IMPFW_CB_MSCO	CL process detects MSCO.
IMPFW_CB_UDIVSBRK	Detects UDI VS's break int.
IMPFW_CB_UDIPSBRK	Detects UDI PS's break int.
IMPFW_CB_DRVERR	Error occurs by IMP Driver.
IMPFW_CB_RTTERR	Error occurs by IMP RTT Driver.
IMPFW_CB_RESOURCEFULL	Error occurs by queue full.

Table 4-5 User behavior of ***e_impfw_callback_reason_t***

Value	Description
IMPFW_CB_STARTED	For debugging using the timing just before CL execution.
IMPFW_CB_TRAP	Execute arbitrary processing after CL execution is completed.
IMPFW_CB_INT	Restart CL with ***R_IMPFW_Resume***.
IMPFW_CB_INT_PBCOVF	Restart CL with ***R_IMPFW_Resume***.
IMPFW_CB_ERROR_ILLEGAL	Check the CL implementation.
IMPFW_CB_ERROR_INTERNAL	(T.B.D.)
IMPFW_CB_USIER	System reset.
IMPFW_CB_INT_SBO0ME	Restart CL with ***R_IMPFW_Resume***.
IMPFW_CB_TRAP_SBO0ME	Execute arbitrary processing after CL execution is completed.
IMPFW_CB_PBCOVF	Wait for CL to complete.
IMPFW_CB_WUPCOV	Wait for CL to complete.
IMPFW_CB_HPINT	Wait for CL to complete.
IMPFW_CB_APIPINT	Wait for CL to complete.
IMPFW_CB_USINT	(Not implement)
IMPFW_CB_END	Wait for CL to complete.
IMPFW_CB_MSCO	Wait for CL to complete.
IMPFW_CB_UDIVSBRK	Notification for debugging.
IMPFW_CB_UDIPSBRK	Notification for debugging.
IMPFW_CB_DRVERR	System reset.
IMPFW_CB_RTTERR	(Not implement)
IMPFW_CB_RESOURCEFULL	Check value of max_queue_num in ***st_impfw_fw_resource_t***

4.3.3 e_impfw_core_type_t

The enumerator is the core type of the IMP-Xn.

```
typedef enum {
            IMPFW_CORE_TYPE_INVALID                         = 0,
            IMPFW_CORE_TYPE_IMP                             ,
            IMPFW_CORE_TYPE_IMP_SLIM                        ,
            IMPFW_CORE_TYPE_OCV                             ,
            IMPFW_CORE_TYPE_DMAC                            ,
            IMPFW_CORE_TYPE_DMAC_SLIM                       ,
            IMPFW_CORE_TYPE_PSCEXE                          ,
            IMPFW_CORE_TYPE_PSCOUT                          ,
            IMPFW_CORE_TYPE_CNN                             ,
            IMPFW_CORE_TYPE_LDMAC                           ,
            IMPFW_CORE_TYPE_DTA
} e_impfw_core_type_t
;
```

Table 4-6 Enumerator of ***e_impfw_core_type_t***

Value	Description
IMPFW_CORE_TYPE_INVALID	Indicates an invalid core
IMPFW_CORE_TYPE_IMP	Core type number of IMP core
IMPFW_CORE_TYPE_IMP_SLIM	Core type number of Slim IMP core
IMPFW_CORE_TYPE_OCV	Core type number of OCV core
IMPFW_CORE_TYPE_DMAC	Core type number of DMAC
IMPFW_CORE_TYPE_DMAC_SLIM	Core type number of Slim DMAC
IMPFW_CORE_TYPE_PSCEXE	Core type number of PSC
IMPFW_CORE_TYPE_PSCOUT	Core type number of PSC output
IMPFW_CORE_TYPE_CNN	Core type number of CNN
IMPFW_CORE_TYPE_LDMAC	Core type is Lock Step DMAC
IMPFW_CORE_TYPE_DTA	Core type number of DTA.

4.3.4 e_impfw_req_priority_t

This enumerator is execution priority value of CL request used with this software.

```
typedef enum {
            IMPFW_REQ_PRIORITY_0                            = 0,
            IMPFW_REQ_PRIORITY_1                            ,
            IMPFW_REQ_PRIORITY_2                            ,
            IMPFW_REQ_PRIORITY_3                            ,
            IMPFW_REQ_PRIORITY_4                            ,
            IMPFW_REQ_PRIORITY_5                            ,
            IMPFW_REQ_PRIORITY_6                            ,
            IMPFW_REQ_PRIORITY_7                            ,
            IMPFW_REQ_PRIORITY_8                            ,
            IMPFW_REQ_PRIORITY_9
} e_impfw_req_priority_t
;
```

Table 4-7 Enumerator of ***e_impfw_req_priority_t***

Value	Description
IMPFW_REQ_PRIORITY_0	CL Request Priority 0 (Lowest)
IMPFW_REQ_PRIORITY_1	CL Request Priority 1
IMPFW_REQ_PRIORITY_2	CL Request Priority 2
IMPFW_REQ_PRIORITY_3	CL Request Priority 3
IMPFW_REQ_PRIORITY_4	CL Request Priority 4
IMPFW_REQ_PRIORITY_5	CL Request Priority 5
IMPFW_REQ_PRIORITY_6	CL Request Priority 6
IMPFW_REQ_PRIORITY_7	CL Request Priority 7
IMPFW_REQ_PRIORITY_8	CL Request Priority 8
IMPFW_REQ_PRIORITY_9	CL Request Priority 9 (Highest)

4.3.5 e_impfw_pmpolicy_t

The enumerator is check type of the power management policy.

```
typedef enum {
            IMPFW_PMPOLICY_PG                               = 1,
            IMPFW_PMPOLICY_CG                               ,
            IMPFW_PMPOLICY_HP
} e_impfw_pmpolicy_t
;
```

Table 4-8 Enumerator of ***e_impfw_pmpolicy_t***

Value	Description
IMPFW_PMPOLICY_PG	Check type of the Power Gated Policy of power management
IMPFW_PMPOLICY_CG	Check type of the Clock Gated Policy of power management
IMPFW_PMPOLICY_HP	Check type of the High-Performance Policy of power management

4.3.6 e_impfw_irq_group_t

The enumerator is the type of IRQ groups.

```
typedef enum {
            IMPFW_IRQ_GROUP_NONE                     = 1,
            IMPFW_IRQ_GROUP_0                        ,
            IMPFW_IRQ_GROUP_1                        ,
            IMPFW_IRQ_GROUP_2
} e_impfw_irq_group_t
;
```

Table 4-9 Enumerator of ***e_impfw_irq_group_t***

Value	Description
IMPFW_IRQ_GROUP_NONE	Does not belong to the interrupt group
IMPFW_IRQ_GROUP_0	Belong to group 0
IMPFW_IRQ_GROUP_1	Belong to group 1
IMPFW_IRQ_GROUP_2	Belong to group 2

4.3.7 e_impfw_instance_t

The enumerator is the type of instance.

```
typedef enum {
            IMPFW_INSTANCE_0                         = 1,
            IMPFW_INSTANCE_1                         ,
            IMPFW_INSTANCE_2                         ,
            IMPFW_INSTANCE_3                         ,
            IMPFW_INSTANCE_4                         ,
            IMPFW_INSTANCE_5                         ,
            IMPFW_INSTANCE_6
} e_impfw_instance_t
;
```

Table 4-10 Enumerator of ***e_impfw_instance_t***

Value	Description
IMPFW_INSTANCE_0	Instance No 0
IMPFW_INSTANCE_1	Instance No 1
IMPFW_INSTANCE_2	Instance No 2
IMPFW_INSTANCE_3	Instance No 3
IMPFW_INSTANCE_4	Instance No 4
IMPFW_INSTANCE_5	Instance No 5
IMPFW_INSTANCE_6	Instance No 6

4.3.8 e_impfw_fatalcode_t

This enumerator shows fatal error information to notify to the user.

```
typedef enum {
        IMPFW_FATALERR_FW_ERROR                      = 0,
        IMPFW_FATALERR_DRV_ERROR                    ,
        IMPFW_FATALERR_UNEXPECT_INT
} e_impfw_fatalcode_t;
```

Table 4-11 Enumerator of ***e_impfw_fatalcode_t***

Value	Description
IMPFW_FATALERR_FW_ERROR	This error is notified when the IMP Framework encounters an unrecoverable error.
IMPFW_FATALERR_DRV_ERROR	This error is notified when the IMP Driver encounters an unrecoverable error.
IMPFW_FATALERR_UNEXPECT_INT	This error occurs unintended interrupt.

4.3.9 e_impfw_interrupt_mask_t

The enumerator is the value of the interrupt mask setting.

```
typedef enum {
        IMPFW_INTERRUPT_UNMASK                       = 0,
        IMPFW_INTERRUPT_MASK                         = 1
} e_impfw_interrupt_mask_t;
```

Table 4-12 Enumerator of ***e_impfw_interrupt_mask_t***

Value	Description
IMPFW_INTERRUPT_UNMASK	Enable interrupt
IMPFW_INTERRUPT_MASK	Disable interrupt

4.3.10 e_impfw_workarea_type_t

The enumerator is the value of the work area type.

```
typedef enum {
        IMPFW_WORKAREA_TYPE_MAIN                     = 0,
        IMPFW_WORKAREA_TYPE_ATTR                     = 1,
        IMPFW_WORKAREA_TYPE_QUE                      = 2
} e_impfw_workarea_type_t
;
```

Table 4-13 Enumerator of ***e_impfw_workarea_type_t***

Value	Description
IMPFW_WORKAREA_TYPE_MAIN	The administrative work area that is used by the IMP Framework.
IMPFW_WORKAREA_TYPE_ATTR	The attribute management work area used by the IMP Framework.

IMPFW_WORKAREA_TYPE_QUE	The work area of the queue management used by the IMP Framework.

4.4 Structure

This section shows the structure used on this software.

4.4.1 st_impfw_core_info_t

This structure shows the core information.

```
typedef struct {
        e_impfw_core_type_t          core_type;

        uint32_t                     core_num;
} st_impfw_core_info_t
;
```

Table 4-14 Structure of ***st_impfw_core_info_t***

Member	in/out	Description
core_type	in	Core type. For supported with SoC, refer to ***IMP Framework Product Information: "2.3.1 The Core number for CL execute", "2.3.2 The core number for CL synchronize execute"***.
core_num	in	Core number in the type. For supported with SoC, refer to ***IMP Framework Product Information: "2.3.1 The number of core".***

Table 4-15 Valid value of ***st_impfw_core_info_t***

Name	Value
core_type	Refer to ***e_impfw_core_type_t*** .
core_num	Refer to ***IMP Framework Product Information: "2.3.1 The number of core"***

4.4.2 st_impfw_version_t

This structure shows the version information of IMP Framework.

```
typedef struct {
        uint32_t                          major;
        uint32_t                          minor;
        uint32_t                          patch;
} st_impfw_version_t
;
```

Table 4-16 Structure of *st_impfw_version_t*

Member	in/out	Description
major	out	The major version number
minor	out	The minor version number
patch	out	The patch version number

Table 4-17 Valid value of *st_impfw_version_t*

Name	Value
major	Refer to *IMPFW_VERSION_MAJOR*
minor	Refer to *IMPFW_VERSION_MINOR*
patch	Refer to *IMPFW_VERSION_PATCH*

4.4.3 st_impfw_initdata_t

This structure shows the initialize information of IMP Framework.

```
typedef struct {
        st_impfw_workarea_info_t          work_area_info[IMPFW_WORKAREA_TYPE_NUM]

        e_impfw_instance_t                instance_num

        uint32_t                          use_core_num;
        st_impfw_core_info_t              *core_info;

        p_impfw_cbfunc_fatal_t (Not       callback_func_fatal;
        implement)

        st_impfw_fw_resource_t            fw_resource;

        st_impfw_drv_resource_t           drv_resource;

        st_impfw_rtt_resource_t (Not      rtt_resource;                    Not implement
        implement)

} st_impfw_initdata_t
;
```

Table 4-18 Structure of ***st_impfw_initdata_t***

Member	in/out	Description
work_area_info[***IMPFW_WORKAREA_TYPE_NUM]***	in	The work area that is used by the IMP Framework. Refer to ***e_impfw_workarea_type_t*** , No.2 of ***User side prepare data*** for the type of workspace information.
instance_num	in	It is the instance number used by IMP Framework. The instance number is a parameter that specifies the interrupt number assigned to IMP-Xn, and the core of *core_info is assigned to this instance number. If you set instance 0, it means that the core specified by * core_info uses the interrupt number to which Instance 0 is assigned. Refer to ***IMP Framework Product Information: "instance number"*** for the correspondence between the instance number and the interrupt number.
use_core_num	in	It is the core number used by IMP Framework. Set the number of cores to initialize. This parameter is the number of cores using at execute. The maximum number that can be specified is the total number of cores supported by the SoC. Refer to ***IMP Framework Product Information: "The number of core"*** for details.
*core_info	in	It is the array of core information used by IMP Framework. Set the core information (core type, core number) that the user initializes. The user must reserve the space specified by use_core_num.
callback_func_fatal	in	It is the pointer to address of fatal notification function.
fw_resource	in	It is the resource used by IMP Framework.
drv_resource	in	It is the resource used by IMP Driver.
rtt_resource	in	It is the resource used by RTT Driver.

Table 4-19 Valid value of ***st_impfw_initdata_t***

Name	Value
work_area_info[***IMPFW_WORKAREA_NUM]***	Refer to ***st_impfw_workarea_info_t***
instance_num	Refer to ***e_impfw_instance_t***
use_core_num	Greater than 1. Refer to ***IMP Framework Product Information: "The number of core"*** for the maximum number.
*core_info	Not NULL
callback_func_fatal	Not NULL
fw_resource	Refer to ***st_impfw_fw_resource_t***
drv_resource	Refer to ***st_impfw_drv_resource_t***
rtt_resource	Refer to ***st_impfw_rtt_resource_t*** (Not implement)

4.4.4 st_impfw_fw_resource_t

This structure shows OSAL resources used by IMP Framework.

```
typedef struct {
            uint32_t                    max_queue_num;
            uint32_t                    max_msg_num;
            osal_mq_id_t                msg_id[IMPFW_MSGTYPE_NUM];
            osal_mutex_id_t             mutex_id[IMPFW_MUTEXTYPE_NUM];
            osal_thread_id_t            task_id[IMPFW_TASKTYPE_NUM];
            osal_milli_sec_t            timeout;
            e_osal_thread_priority_t    task_priority;
            size_t                      task_stacksize;
} st_impfw_fw_resource_t
;
```

Table 4-20 Structure of ***st_impfw_fw_resource_t***

Member	in/out	Description
max_queue_num	in	Maximum number of queues to reserve for specified all cores by ***R_IMPFW_Init***. (Same as the number of ***Attribute Handle***.) Maximum number of requests that the user wants to queue into the IMP Framework.
max_msg_num	in	Maximum number of message queues to reserve for specified all cores by ***R_IMPFW_Init***. Maximum number of messages to use within the IMP Framework. The recommended value is use_core_num * 3.
msg_id[***IMPFW_MSGTYPE_NUM***]	in	Message ID used by IMP Framework
mutex_id[***IMPFW_MUTEXTYPE_NUM]***	in	Mutex ID used by IMP Framework
task_id[***IMPFW_TASKTYPE_NUM***]	in	Task ID used by the IMP Framework
task_priority	in	Priority of tasks used by the IMP Framework
timeout	in	Mutex or message timeout value
task_stacksize	in	Stack size of tasks used by IMP Framework

Table 4-21 Valid value of ***st_impfw_fw_resource_t***

Name	Value
max_queue_num	1～ The maximum value depends on the size reserved in the Management memory area. Refer to ***User side prepare data***.
max_msg_num	1～ The maximum value depends on OSAL resources. Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** and ***Operating System Abstraction Layer API(OSAL API)***
msg_id[***IMPFW_MSGTYPE_NUM***]	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** and ***Operating System Abstraction Layer API(OSAL API)***
mutex_id[***IMPFW_MUTEXTYPE_NUM***]	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***

task_id[*IMPFW_TASKTYPE_NUM*]	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***
task_priority	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***
timeout	Greater than equal to 0 Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***
task_stacksize	Refer to ***IMP Framework Product Information: "Memory Requirement"*** about the valid stack size

4.4.5 st_impfw_drv_resource_t

This structure shows OSAL resources used by IMP Driver.

```
typedef struct {
        osal_mutex_id_t                    mutex_id;
        osal_milli_sec_t                   mutex_timeout;
        e_osal_interrupt_priority_t        int_priority;
} st_impfw_drv_resource_t
;
```

Table 4-22 Structure of ***st_impfw_drv_resource_t***

Member	in/out	Description
mutex_id	in	Mutex ID used in IMP Driver
mutex_timeout	in	Mutex timeout value
int_priority	in	Interrupt priority

Table 4-23 Valid value of ***st_impfw_drv_resource_t***

Name	Value
mutex_id	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***
mutex_timeout	Greater than equal 0 Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***
int_priority	Refer to ***Operating System Abstraction Layer(OSAL) API Application Note*** *and* ***Operating System Abstraction Layer API(OSAL API)***

4.4.6 st_impfw_rtt_resource_t (Not implement)

This structure shows Runtime test resources used by IMP RTT Driver.

4.4.7 st_impfw_rtt_info_t (Not implement)

This structure shows Runtime test information used by IMP RTT Driver.

4.4.8 st_impfw_interrupt_info_t

This structure shows interrupt information used by IMP Framework.

```
typedef struct {
        e_impfw_irq_group_t                irq_group;

        uint32_t                           group_core_num;
        st_impfw_core_info_t               *group_core_info;

        st_impfw_interrupt_mask_t          interrupt_mask;
} st_impfw_interrupt_info_t
;
```

Table 4-24 Structure of ***st_impfw_interrupt_info_t***

Member	in/out	Description
irq_group	in	IRQ group information. Set up the interrupt groups to register. The core that can be set for an interrupt group depends on the SoC. Refer to ***IMP Framework Product Information: "IRQ group"***
group_core_num	in	Number of cores to be grouped. Specify 0 to cancel the group.
*group_core_info	in	Core information to be grouped. Set the all core information (core type, core number) to be set in the interrupt group. The user must reserve the space specified by “group_core_num”.
interrupt_mask	in	Each interrupt mask information. The values for all members of this structure are set. The initial setting of the interrupt mask is that TRAP and INT interrupts are valid and unmasked, the other interrupts are masked.

Table 4-25 Valid value of ***st_impfw_interrupt_info_t***

Name	Value
irq_group	Refer to ***IMP Framework Product Information: "IRQ group"***
group_core_num	The maximum number of cores that can be registered in the same group depends on the SoC. Refer to ***IMP Framework Product Information: "The number of maximum core"***.
*group_core_info	Not NULL (group_core_num is greater than 0). NULL (group_core_num is 0)
interrupt_mask	Refer to ***st_impfw_interrupt_mask_t***

4.4.9 st_impfw_interrupt_mask_t

This structure shows each interrupt mask setting used by IMP Framework. Supported interrupts depend on the SoC. Refer to ***IMP Framework Product Information: "Interrupt type"*** for details.

```
typedef struct {
        e_impfw_interrupt_mask_t        end_mask;
        e_impfw_interrupt_mask_t        wupcovf_mask;
        e_impfw_interrupt_mask_t        usier_mask;
        e_impfw_interrupt_mask_t        usint_mask;
        e_impfw_interrupt_mask_t        pbcovf_mask;
        e_impfw_interrupt_mask_t        sbo0me_mask;
        e_impfw_interrupt_mask_t        hpint_mask:
        e_impfw_interrupt_mask_t        apipint_mask;
        e_impfw_interrupt_mask_t        msco_mask;
} st_impfw_interrupt_mask_t;
```

Table 4-26 Structure of ***st_impfw_interrupt_mask_t***

Member	in/out	Description
end_mask	in	The END interrupt mask setting
wupcovf_mask	in	The WUPCOVF interrupt mask setting
usier_mask	in	The USIER interrupt mask setting
usint_mask	in	The USINT interrupt mask setting
pbcovf_mask	in	The PBCOVF interrupt mask setting
sbo0me_mask	in	The SBO0ME interrupt mask setting
hpint_mask	in	The HPINT interrupt mask setting
apipint_mask	in	The APIPINT interrupt mask setting
msco_mask	in	The MSCO interrupt mask setting

Table 4-27 Valid value of ***st_impfw_interrupt_mask_t***

Name	Value
end_mask	Refer to ***e_impfw_interrupt_mask_t***
wupcovf_mask	Refer to ***e_impfw_interrupt_mask_t***
usier_mask	Refer to ***e_impfw_interrupt_mask_t***
usint_mask	Refer to ***e_impfw_interrupt_mask_t***
pbcovf_mask	Refer to ***e_impfw_interrupt_mask_t***
sbo0me_mask	Refer to ***e_impfw_interrupt_mask_t***
hpint_mask	Refer to ***e_impfw_interrupt_mask_t***
apipint_mask	Refer to ***e_impfw_interrupt_mask_t***
msco_mask	Refer to ***e_impfw_interrupt_mask_t***

4.4.10 st_impfw_workarea_info_t

This structure provides information about the work area that is used by the IMP framework.

```
typedef struct {
        void                *p_work_addr;
        uint32_t            work_size;
} st_impfw_workarea_info_t
;
```

Table 4-28 Structure of ***st_impfw_workarea_info_t***

Member	in/out	Description
*p_work_addr	in	It is the pointer to address of ***Management memory area***. The ***Management memory area*** must be placed on an 8-byte boundary.
work_size	in	It is the size of ***Management memory area***.

Table 4-29 Valid value of ***st_impfw_workarea_info_t***

Name	Value
*p_work_addr	Not NULL.
work_size	Refer to No.2 in ***Table 3-1***

5. Function

The lifetime of parameter not described for lifetime is until the function returns.

5.1 Function List

5.1.1 External Function

This section shows the external functions in ***Table 5-1***. And executable state of each function is shown in the specification of each function.

Table 5-1 List of External Functions

Function Name	Description	Mandatory Function
R_IMPFW_Init	Initialize this software.	Mandatory
R_IMPFW_Execute	Execute the CL processing.	Mandatory
R_IMPFW_Quit	Quit this software.	Mandatory
R_IMPFW_Resume(Not implement)	Resume the CL processing.	Mandatory
R_IMPFW_SetPmPolicy(Not implement)	Set PM policy.	Mandatory
R_IMPFW_GetVersion	Get IMP Framework Version.	Mandatory
R_IMPFW_AttrInit	Initialize attribute parameter.	Mandatory
R_IMPFW_AttrSetCl	Set attributes rerated to CL execution.	Mandatory
R_IMPFW_AttrSetPair	Set attributes rerated to pair function.	Optional
R_IMPFW_AttrSetCoremap	Set attributes rerated to core map function.	Optional
R_IMPFW_AttrSetInterrupt(Not implement)	Set attributes rerated to interrupt function.	Optional

5.1.2 Callback Function

This section shows the Callback functions in ***Table 5-2***.

Table 5-2 List of Callback Functions

Function Name	Description
p_impfw_cbfunc_t	Execute the callback function when IMP-Xn/Field BIST processing starts, ends, and an error occurs.

Function Name	Description
p_impfw_cbfunc_fatal_t (Not implement)	Execute the callback function when notifying a fatal error.

5.2 External Function Prototypes

5.2.1 R_IMPFW_Init

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_Init
(
        const st_impfw_initdata_t
                *const   p_initdata,
        impfw_ctrl_handle_t                *const   p_handle
 ) ;
```

<Input Parameters>

Parameter	Description
*p_initdata	The pointer to the ***st_impfw_initdata_t*** structure. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

Parameter	Description
*p_handle	The pointer to the ***impfw_ctrl_handle_t***. The lifetime of this parameter is the period from the ***R_IMPFW_Init*** is executed until ***R_IMPFW_Quit*** is executed.

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(***p_impfw_cbfunc_t***
)

<Executable State(all cores)>

Library State	Permission
Uninitialized State	■Permitted / □Prohibited
Ready State	□Permitted / ■Prohibited
Wait State	□Permitted / ■Prohibited
Execute State	□Permitted / ■Prohibited
INT State	□Permitted / ■Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_DRVERROR

IMPFW_EC_NG_NOTSUPPORT

IMPFW_EC_NG_RESOURCEFULL

IMPFW_EC_NG_TIMEOUT

<Description>

This API is IMP Framework initialization. Execute this function before using the IMP Framework. If the IMP Framework is executed without initializing by ***R_IMPFW_Init***
, operation is not guaranteed.

1. The IMP Framework is launched from the User Application, and initialization. It receives a pointer to each data (the information of ***Management memory area***, instance number, the core information to use, the resource information of IMP Framework, IMP Driver, and IMP RTT Driver, etc.) by argument (p_initdata). It also receives ***Control Handle*** as an argument(p_handle). After that, it will be used to refer to the work memory in the internal processing of IMP Framework until ***R_IMPFW_Quit***
2. is executed.
3. Initialize ***Management memory area***. It also creates resources (OSAL resources and queues) from pointers to relevant data in the IMP Framework resource information.
4. It passes a pointer to the relevant data to the IMP Driver, and calls the driver's API to initialize it.
5. The IMP Framework becomes ***Ready State***
6. and returns to the User Application.

For the status, see the state machine.

<Notes>

None.

5.2.2 R_IMPFW_Execute

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_Execute
(
        impfw_ctrl_handle_t                          handle,
        const st_impfw_core_info_t
                                    *const p_core_info,
        const impfw_attr_handle_t                    attrhandle,
        p_impfw_cbfunc_t
                                    callback_func,
        void                                         *const p_callback_args
        );
```

<Input Parameters>

Parameter	Description
handle	The lifetime of this parameter is the period from the *R_IMPFW_Init* is executed until *R_IMPFW_Quit* is executed.
*p_core_info	The pointer to the information about the core that performs CL Execution. The core information should be the same as the core information set by the argument p_core_info in *R_IMPFW_AttrInit*. The lifetime of this parameter is until this function returns.
attrhandle	The lifetime of this parameter is the period from the *R_IMPFW_AttrInit* is executed until *R_IMPFW_Quit* is executed.
callback_func	The pointer to the *p_impfw_cbfunc_t* function. Refer to *p_impfw_cbfunc_t* . The lifetime of this parameter is until the completion of CL execution is notified by callback function.
*p_callback_args	The pointer to the additional data for callback function. The lifetime of this parameter is until the completion of CL execution is notified by callback function.

<Input-Output Parameters>
None.

<Output Parameters>
None.

<Function Attribute>

Attributes	Value
Categories	□Synchronous function / ■Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_t*)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited

Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_ATTRERROR

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

IMPFW_EC_NG_DRVERROR

<Description>

This API is a CL execution function of IMP Framework.
Setting of IMP-Xn processing from the specified core information and CL.

CL is executed according to the request parameter associated with ***Attribute Handle.***

When this function is executed, CL is added to Queue management.
At this time, if the target core is stopped, added CL is executed. Even if the not target core is executing CL or Runtime Test, it is executed. However, if the target core is executing CL or Runtime Test, it will not be executed in this function.

Completion of IMP-Xn processing is notified to the user application by calling the ***p_impfw_cbfunc_t*** function specified by callback function. By ***R_IMPFW_Execute*** or ***R_IMPFW_Resume,*** the callback is executed at least once for each the interrupt.
Until this notification is returned, the request parameter associated with ***Attribute Handle*** cannot be changed. (That is, R_IMPFW_AttrSet***(***R_IMPFW_AttrSetCl*** etc..) cannot be executed.)
If you want to change the attribute for the next ***R_IMPFW_Execute*** before the callback is returned,
you need to execute ***R_IMPFW_AttrInit*** separately and prepare another handle.

1. The IMP Framework is executed from the User Application and execute processing. IMP Framework receives the information required for CL execution (the physical address of CL etc..) through ***Attribute Handle***.
2. The input CL is stored in Queue.
3. If there is no running of the target core, processing of the IMP-Xn is started through the IMP Driver.
4. IMP-Xn will execute by IMP Driver.
5. It will receive return value from IMP Driver.
6. The IMP Framework returns to the User Application.

IMPFW_CB_TRAP, ***IMPFW_CB_INT***, ***IMPFW_CB_ERROR_ILLEGAL***(IER interrupt) and ***IMPFW_CB_WUPCOV*** are notified even if ***R_IMPFW_AttrSetInterrupt*** is not executed.

For the status, see the state machine.

<Caution>

If the INT instruction is detected during CL processing, the INT state must be cleared with the ***R_IMPFW_Resume***(Not implement) function to resume CL

- ***p_impfw_cbfunc_t***
- function (User Application)

  The ***p_impfw_cbfunc_t*** function specified in callback_func is called when CL processing is started and completed. If the IMP-Xn detects an illegal command in CL processing, an error code is returned to callback_func after suspending CL processing. Refer to ***p_impfw_cbfunc_t*** ***.***

<Notes>

None

5.2.3 R_IMPFW_Quit

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_Quit
(
         impfw_ctrl_handle                    handle
         ) ;
```

<Input Parameters>

Parameter	Description
handle	The lifetime of this parameter is the period from the ***R_IMPFW_Init*** is executed until ***R_IMPFW_Quit*** is executed.

<Input-Output Parameters>

None

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(***p_impfw_cbfunc_t***
)

<Executable State(all cores)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	□Permitted / ■Prohibited
Execute State	□Permitted / ■Prohibited
INT State	□Permitted / ■Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_DRVERROR

IMPFW_EC_NG_TIMEOUT

<Description>

This API is IMP Framework completion process. This function must be executed when all core of IMP-Xn is not execute processing (***Ready State***
), otherwise returns error. Please check that all processing is completed before executing.

1. The IMP Framework is executed from the user application and completion processing is execute. It receives ***Control Handle*** as an argument(handle).
2. It passes a pointer to the relevant data to the IMP Driver, and calls the driver's API to quit it.
3. Clear ***Management memory area***. And releases the resources (OSAL, Queue) being used.
4. The IMP Framework becomes ***Uninitialized State***
5. and returns to the User Application.

For the status, see the state machine.

<Notes>

None.

5.2.4 R_IMPFW_Resume(Not implement)

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_Resume(Not implement)
(
        impfw_ctrl_handle                    handle,
        const st_impfw_core_info_t
                 *const   p_core_info
        ) ;
```

<Input Parameters>

Parameter	Description
handle	The lifetime of this parameter is the period from the ***R_IMPFW_Init*** is executed until ***R_IMPFW_Quit*** is executed.
*p_core_info	Core number for clearing the INT interrupt state. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

None

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Categories	□Synchronous function / ■Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	■Permitted / □Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(***p_impfw_cbfunc_t***)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	□Permitted / ■Prohibited
Wait State	□Permitted / ■Prohibited
Execute State	□Permitted / ■Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API has clear to the INT interrupt state, and CL execution resumes. If it is executed in a state other than the

INT State
, return error (***IMPFW_EC_NG_SEQSTATE***).

1. The IMP Framework is executed from the user application and receive the resume instruction from User Application.
2. It passes a pointer to the relevant data to the IMP Driver, and calls the driver's API (Resume).
3. IMP-Xn will execute by IMP Driver.
4. It will receive return value from IMP Driver.
5. The IMP Framework returns to the User Application.

For the status, see the state machine.

<Notes>
None.

5.2.5 R_IMPFW_SetPmPolicy(Not implement)

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_SetPmPolicy(Not implement)
(
        impfw_ctrl_handle_t                 handle,
        const st_impfw_core_info_t
                 *const p_core_info,
        const e_impfw_pmpolicy_t
                 policy
        );
```

<Input Parameters>

Parameter	Description
handle	The lifetime of this parameter is the period from the ***R_IMPFW_Init*** is executed until ***R_IMPFW_Quit*** is executed.
*p_core_info	The core kind info that is executed Runtime Test. The lifetime of this parameter is until this function returns.
policy	The power management policy. Refer to ***e_impfw_pmpolicy_t*** .

<Input-Output Parameters>

None.

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	□Synchronous function / ■Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes / □No

(*)"callback" is IMPFW callback(***p_impfw_cbfunc_t***)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API sets the power management policy of the specified core.

<Notes>

5.2.6 R_IMPFW_GetVersion

<Function Prototypes>

```
const st_impfw_version_t
* const R_IMPFW_GetVersion
(
        void
        );
```

<Input Parameters>

None.

<Input-Output Parameters>

None.

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	■Permitted / □Prohibited
Call from callback(*)	■Permitted / □Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	□Yes(see Executable State) / ■No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_t*)

<Executable State(target core)>

Library State	Permission
Uninitialized State	■Permitted / □Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

The pointer of structure of *st_impfw_version_t*

<Description>

This API return the version of IMP Framework.

<Notes>

None.

5.2.7 R_IMPFW_AttrInit

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_AttrInit
(
        const impfw_ctrl_handle_t                  handle
        const st_impfw_core_info_t
                *const p_core_info
        impfw_attr_handle_t                        *const p_attrhandle
        );
```

<Input Parameters>

Parameter	Description
handle	The lifetime of this parameter is the period from the ***R_IMPFW_Init*** is executed until ***R_IMPFW_Quit*** is executed.
*p_core_info	The pointer to the core information associated with ***Attribute Handle***. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

Parameter	Description
*p_attrhandle	The lifetime of this parameter is the period from the ***R_IMPFW_AttrInit*** is executed until ***R_IMPFW_Quit*** is executed

<Output Parameters>

None

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(***p_impfw_cbfunc_t***)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_RESOURCEFULL

IMPFW_EC_NG_TIMEOUT

<Description>

This API initializes ***Attribute Handle*** associated with the request to the IMP Framework. ***Attribute Handle*** is initialized by associating it with the IMP-Xn core (p_core_info).

By specifying ***Attribute Handle*** initialized here in the argument of R_IMPFW_AttrSet***(***R_IMPFW_AttrSetCl*** etc..), the request parameter to IMP Framework can be set in association with ***Attribute Handle***.

When executing multiple CL processes without waiting for the CL process completion notification from the IMP-Xn cores, it is necessary to call this API as many times as there are requests in order to prepare ***Attribute Handle*** corresponding to each process.

<Notes>

None.

5.2.8 R_IMPFW_AttrSetCl

<Function Prototypes>

e_impfw_api_retcode_t
R_IMPFW_AttrSetCl
(
const *impfw_attr_handle_t* attrhandle
uintptr_t claddr_phys
uint32_t clsize
e_impfw_req_priority_t
priority
);

<Input Parameters>

Parameter	Description
attrhandle	The lifetime of this parameter is the period from the *R_IMPFW_AttrInit* is executed until *R_IMPFW_Quit* is executed.
claddr_phys	The physical address of CL that is executed by each core. The value should be 4-byte aligned. The lifetime of this parameter is until this function returns. However, the memory area pointed to by this parameter has following lifetime. The lifetime of memory area is until the completion of CL execution is notified by callback function.
clsize	Size of CL. The lifetime of this parameter is until this function returns.
priority	The priority for the CL execution. Do not set different priority if the same pair_id is set among pair cores. The value shall be in range of *IMPFW_REQ_PRIORITY_0* to *IMPFW_REQ_PRIORITY_9*. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

None.

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_t*
)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited

INT State	■Permitted / □Prohibited

<Event Notification>
None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API sets the request parameters (CL information) associated with ***Attribute Handle***.

When ***R_IMPFW_Execute***

is called, this request parameter is referenced through ***Attribute Handle*** and is used in CL execution.

Until the completion of CL execution is notified by the callback, the request parameter associated for the same ***Attribute Handle*** cannot be changed by this API.(In this case, ***IMPFW_EC_NG_SEQSTATE*** is returned.)

See the description of ***R_IMPFW_Execute***

for details.

For CL setting, the kind of cores shall be set to the argument, core and the physical memory address stored the CL information for the core shall be set to the argument, claddr_phys. For the claddr_phys is specify the memory address secured by 4-byte alignment. If it is not 4-byte alignment, return error.

<Notes>
See ***Appendix 1.Priority Function*** for details on the priority.

5.2.9 R_IMPFW_AttrSetPair

<Function Prototypes>

```
e_impfw_api_retcode_t
 R_IMPFW_AttrSetPair
(
            const impfw_attr_handle_t                attrhandle,
            uint32_t   pair_id,
            uint32_t   pair_num,
            const st_impfw_core_info_t
            *const    pair_core_info
            );
```

<Input Parameters>

Parameter	Description
attrhandle	The lifetime of this parameter is the period from the *R_IMPFW_AttrInit* is executed until *R_IMPFW_Quit* is executed.
pair_id	The id of the CL to be executed in pairs When “pair_num” parameter is 0, this parameter is ignored.
pair_num	The numbers of the CL to be executed in pairs. The maximum number is (Not implement).

*pair_core_info	The core of the CL to be executed in pairs.
	When “pair_num” parameter is 0, this parameter is ignored.
	The usage example is shown below.
	st_impfw_core_info_t *pair_core_info[pair_num];
	/* The length of the array is determined by the argument pair_num. */
	st_impfw_core_info_t core0;
	st_impfw_core_info_t core1;
	: :
	pair_core_info[0] = &(core0);
	pair_core_info[1] = &(core1);
	The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

None.

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)”callback” is IMPFW callback(*p_impfw_cbfunc_t*
)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited

Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>
None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API sets the request parameters (Pair core information) associated with ***Attribute Handle***.

When ***R_IMPFW_Execute***

is called, this request parameter is referenced through ***Attribute Handle*** and is used in CL execution.

Until the completion of CL execution is notified by the callback, the request parameter associated for the same ***Attribute Handle*** cannot be changed by this API.(In this case, ***IMPFW_EC_NG_SEQSTATE*** is returned.)

See the description of ***R_IMPFW_Execute***

for details.

This API is Optional. (See ***Table 5-1***)

If this API is not executed, ***R_IMPFW_Execute***

will be processed with the following default values. And to cancel the pair, set the following values.

pair_num = 0

pair_id = 0

pair_core_info = NULL

The pair ID can be reused after the CL execution using the corresponding pair ID is completed (after the completion callback is executed).Refer to ***2.Pair Function*** for details.

<Notes>

See ***Appendix 2.Pair Function*** for details on the pair function.
The pair_id is specified in the argument. This is used to synchronize specific cores with each other. Set the same ID for the cores to be synchronized, and set the number of synchronous cores to pair_num.

5.2.10 R_IMPFW_AttrSetCoremap

<Function Prototypes>

e_impfw_api_retcode_t
R_IMPFW_AttrSetCoremap
(
const *impfw_attr_handle_t* attrhandle
const *st_impfw_core_info_t*
coremap[*IMPFW_COREMAP_MAXID*]
);

<Input Parameters>

Parameter	Description
attrhandle	The lifetime of this parameter is the period from the *R_IMPFW_AttrInit* is executed until *R_IMPFW_Quit* is executed.
coremap[*IMPFW_COREMAP_MAXID*]	The core mapping array that is set to CORE number for WUP/SLP command to synchronize between cores. (For the detail of core map, please refer to *4.Setting CoreMap)*. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>
None.

<Output Parameters>
None.

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_t*
)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>
None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API sets the request parameters (coremap information) associated with ***Attribute Handle***.

When ***R_IMPFW_Execute*** is called, this request parameter is referenced through ***Attribute Handle*** and is used in CL execution.

Until the completion of CL execution is notified by the callback, the request parameter associated for the same ***Attribute Handle*** cannot be changed by this API.(In this case, ***IMPFW_EC_NG_SEQSTATE*** is returned.)

See the description of ***R_IMPFW_Execute*** for details.

This API is Optional. (See ***Table 5-1***)

If this API is not executed, ***R_IMPFW_Execute*** will be processed with the following default values. And to cancel the coremap, set the following values.

core_map[0 .. ***IMPFW_COREMAP_MAXID-1***].core_type = IMPFW_CORE_TYPE_INVALID
core_map[0 .. ***IMPFW_COREMAP_MAXID-1***].core_num = 0

If the WUP / SLP instruction is not used for CL, it is not necessary to call this API. coremap is ignored.

However, when using WUP / SLP instructions, it is necessary to call this API. If this API is not called in this case, the operation cannot be guaranteed.

<Notes>

See ***Appendix 4.Setting CoreMap*** for details on the pair function.

5.2.11 R_IMPFW_AttrSetInterrupt(Not implement)

<Function Prototypes>

e_impfw_api_retcode_t
R_IMPFW_AttrSetInterrupt(Not implement)
(
const *impfw_attr_handle_t* attrhandle,
const *st_impfw_interrupt_info_t*
*const p_interrupt_info
);

<Input Parameters>

Parameter	Description
attrhandle	The lifetime of this parameter is the period from the *R_IMPFW_AttrInit* is executed until *R_IMPFW_Quit* is executed.
*p_interrupt_info	The pointer to the *st_impfw_interrupt_info_t* structure. The lifetime of this parameter is until this function returns.

<Input-Output Parameters>

None.

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	■Synchronous function / □Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*)	□Permitted / ■Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_t*
)

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	■Permitted / □Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

IMPFW_EC_OK

IMPFW_EC_NG_PARAM

IMPFW_EC_NG_SYSTEMERROR

IMPFW_EC_NG_SEQSTATE

IMPFW_EC_NG_TIMEOUT

<Description>

This API sets the request parameters (interrupt information) associated with ***Attribute Handle***.

When ***R_IMPFW_Execute***

is called, this request parameter is referenced through ***Attribute Handle*** and is used in CL execution.

Until the completion of CL execution is notified by the callback, the request parameter associated for the same ***Attribute Handle*** cannot be changed by this API.(In this case, ***IMPFW_EC_NG_SEQSTATE*** is returned.)

The group function uses this API and ***R_IMPFW_AttrSetPair*** together.

A setting example for grouping IMP Core 0 and IMP Core 1 into IMPFW_IRQ_GROUP_0 is shown below.

interrupt_info-> irq_group = IMPFW_IRQ_GROUP_0

interrupt_info-> group_core_num = 2

interrupt_info-> group_core_info = { { IMPFW_CORE_TYPE_IMP, 0 }, { IMPFW_CORE_TYPE_IMP , 1} }

interrupt_info-> interrupt_mask = Set all interrupt to ***IMPFW_INTERRUPT_MASK***

In addition, in the group function, the callback function is executed for the number of cores registered in the group. For example, if you group IMP Core0 and IMP Core1, the callback will be executed twice with the execution result of IMP Core0 and IMP Core1. The user should perform the process according to the contents of the callback reason.

Refer to ***e_impfw_callback_reason_t*** for callback reason. And Refer to ***3.IRQ Group function*** for group function sequence.

See the description of ***R_IMPFW_Execute***

for details.

This API is Optional. (See ***Table 5-1***)

If this API is not executed, ***R_IMPFW_Execute***

will be processed with the following default values. And to cancel the group, set the following values.

interrupt_info-> irq_group = IMPFW_IRQ_GROUP_NONE

interrupt_info-> group_core_num = 0

interrupt_info-> group_core_info = NULL

interrupt_info-> interrupt_mask = 0

<Notes>

None.

5.3 Callback Function Prototypes

5.3.1 p_impfw_cbfunc_t

<Function Prototypes>

```
int32_t (* p_impfw_cbfunc_t
  (
e_impfw_callback_reason_t
                                            reason,
        const st_impfw_core_info_t
                 *const p_core_info,
        int32_t                                     add_info,
        void                                        *const p_user_arg
        );
```

<Input Parameters>

Parameter	Description
reason	This indicates the reason for the call.
p_core_info	This indicates the type and number of the target core.
add_info	This is additional information on the call reason.
p_user_arg	The "callback_args" specified on ***R_IMPFW_Execute*** .

<Input-Output Parameters>

None

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	□Synchronous function / ■Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*1)	□Permitted / □Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	□Permitted / ■Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

Return code	Description
0	Successful
Not 0	Callback function execute error

<Description>

This API is a callback function of User Application.

Please do not call the function directly from within the callback function specified by callback_func.
Please Implement so that the processing time of this callback function is the shortest because it affects the executing CL / RTT time.

An argument to the ***p_impfw_cbfunc_t***
function changes by the following conditions.

For reason :
- If the CL processing is started on IMP-Xn, ***IMPFW_CB_STARTED*** is given to the argument.
- If the CL processing is completed normally, ***IMPFW_CB_TRAP*** is given to the argument.
- If the CL process completes with illegal command, ***IMPFW_CB_ERROR_ILLEGAL*** is given as an argument.
- If the CL process completes with INT command state, ***IMPFW_CB_INT*** is given as an argument.
- If the CL process completes with INT command state and Performance Busy Counter Overflow, ***IMPFW_CB_INT_PBCOVF*** is given as an argument.

(For the INT command, Refer to ***4.3.2***.)
- If the USIER error detected, ***IMPFW_CB_USIER*** is given to the argument. (*2)
- If the SBO0ME error detected at INT, ***IMPFW_CB_INT_SBO0ME*** is given to the argument.
- If the SBO0ME error detected at TRAP, ***IMPFW_CB_TRAP_SBO0ME*** is given to the argument.

For p_core_info :
- The target core information of CL processing or Safety Mechanisms.

For add_info :
- If the CL process completes with TRAP or INT command state, TRAP or INT code (8 bit) is given as an argument. Otherwise -1 is given.

(For the TRAP and INT command, Refer to ***4.3.2***.)

For p_user_arg :
- The "callback_args" specified by ***R_IMPFW_Execute***
.

<Notes>
(*1) Out of scope because this API itself is callback.
(*2) When this callback reason is returned, execute system reset and reconsider the CL.

5.3.2 p_impfw_cbfunc_fatal_t (Not implement)

<Function Prototypes>

```
void p_impfw_cbfunc_fatal_t (Not implement)
  (
        e_impfw_fatalcode_t         error,
        uint32_t                    code
        );
```

<Input Parameters>

Parameter	Description
error	Error code.
code	Error Description

<Input-Output Parameters>

None

<Output Parameters>

None.

<Function Attribute>

Attributes	Value
Categories	□Synchronous function / ■Asynchronous function
Call from interrupt	□Permitted / ■Prohibited
Call from callback(*1)	□Permitted / □Prohibited
Reentrant	■Permitted / □Prohibited
State restriction	■Yes(see Executable State) / □No

(*)"callback" is IMPFW callback(*p_impfw_cbfunc_fatal_t* (Not implement))

<Executable State(target core)>

Library State	Permission
Uninitialized State	□Permitted / ■Prohibited
Ready State	■Permitted / □Prohibited
Wait State	□Permitted / ■Prohibited
Execute State	■Permitted / □Prohibited
INT State	■Permitted / □Prohibited

<Event Notification>

None.

<Return Codes>

None

<Description>

You will be notified when an unrecoverable error occurs.
Perform a system reset.

<Notes>

Appendix

1.Priority Function

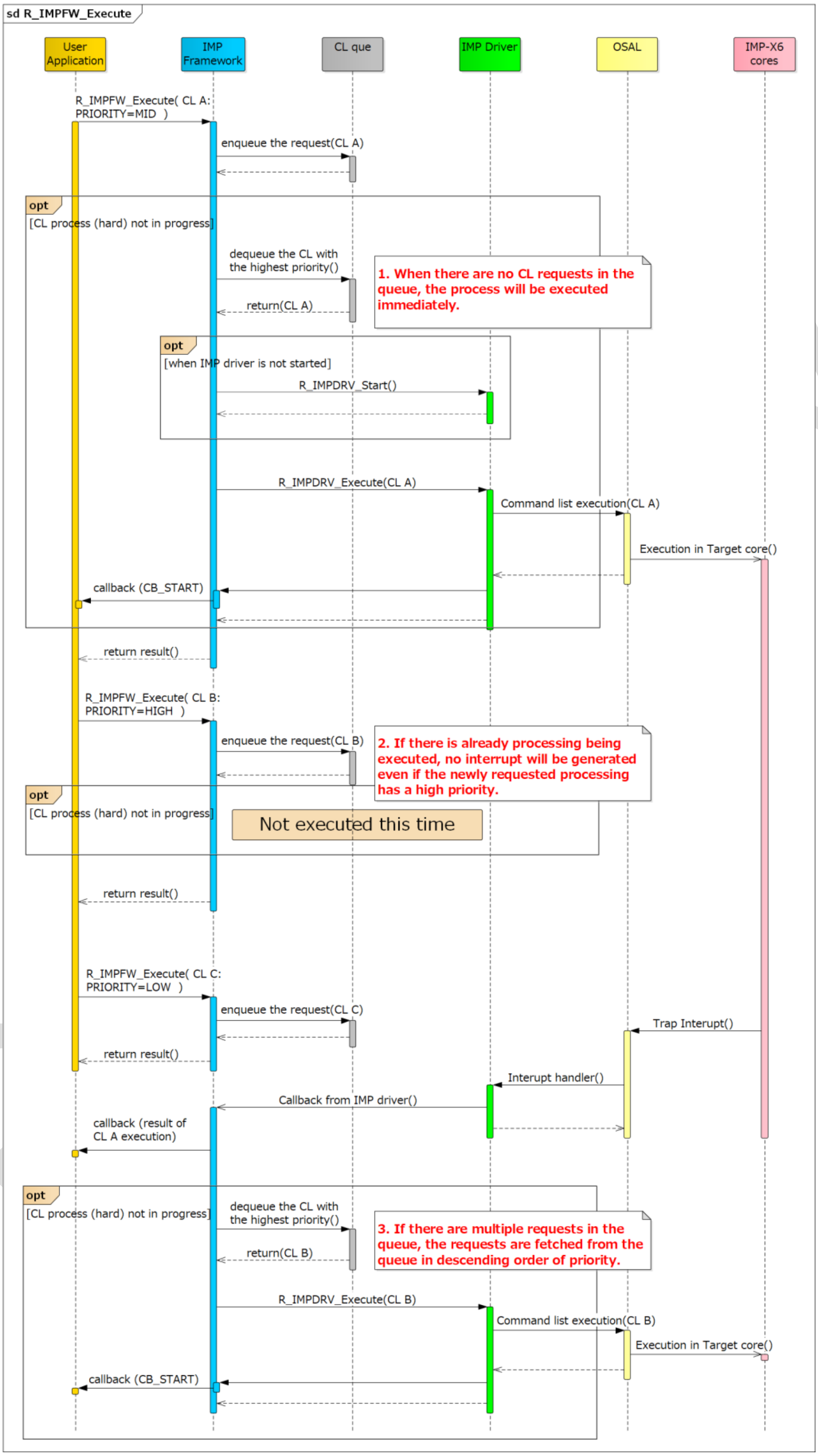
This section describes the request priority function.

The request priority function is a parameter that controls the execution order of CL and Runtime Test.

The priority function has the following three features.

1. When there are no CL requests in the queue, the process will be executed immediately.
2. If there is already processing being executed, no interrupt will be generated even if the newly requested processing has a high priority.
3. If there are multiple requests in the queue, the requests are fetched from the queue in descending order of priority.

The request priority function-flow is shown in ***Figure1-1 Priority Function***.


sd R_IMPFW_Execute
User Application
IMP Framework
CL que
IMP Driver
OSAL
IMP-X6 cores
R_IMPFW_Execute(CL A: PRIORITY=MID)
enqueue the request(CL A)
opt
[CL process (hard) not in progress]
dequeue the CL with the highest priority()
return(CL A)
opt
[when IMP driver is not started]
R_IMPDRV_Start()
R_IMPDRV_Execute(CL A)
Command list execution(CL A)
Execution in Target core()
callback (CB_START)
return result()
R_IMPFW_Execute(CL B: PRIORITY=HIGH)
enqueue the request(CL B)
opt
[CL process (hard) not in progress]
Not executed this time
return result()
R_IMPFW_Execute(CL C: PRIORITY=LOW)
enqueue the request(CL C)
Trap Interupt()
return result()
Interupt handler()
Callback from IMP driver()
callback (result of CL A execution)
opt
[CL process (hard) not in progress]
dequeue the CL with the highest priority()
return(CL B)
R_IMPDRV_Execute(CL B)
Command list execution(CL B)
Execution in Target core()
callback (CB_START)

1. When there are no CL requests in the queue, the process will be executed immediately.

2. If there is already processing being executed, no interrupt will be generated even if the newly requested processing has a high priority.

3. If there are multiple requests in the queue, the requests are fetched from the queue in descending order of priority.

Figure1-1 Priority Function

2.Pair Function

This section describes the pair function. Use the pair function with the following settings.

CL processing between cores can be synchronized by pair function.

Table A-1 Parameter in ***R_IMPFW_AttrSetPair***

Pair Parameter	description
pair_core_info	In this parameter, specify the core to be paired with.
pair_num	In this parameter, specify the number of cores to be paired with.
pair_id	In this parameter, specify the ID for determining the combination of pairs.

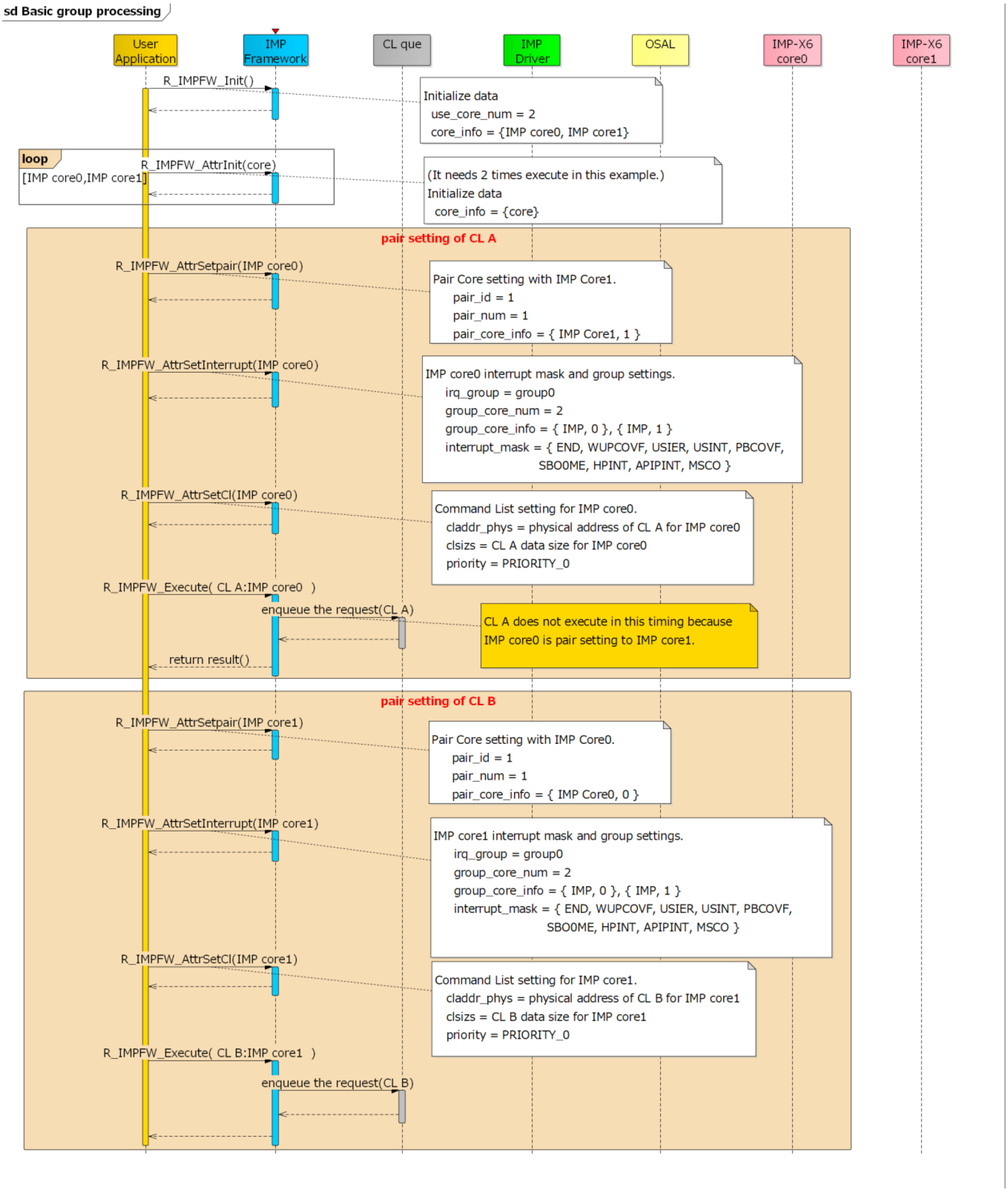
These parameters are used in the ***R_IMPFW_Execute*** function.

The pair function has the following two features.

1. IMP Framework does not start running pair processing until all the pair CLs or all the Runtime Test requests with same ID are set.
   A description of this feature is shown in ***Figure2-1 Pair Function(Basic pair processing)***.

2. If two pairs of requests are requested in parallel, the complete pair of requests will be executed first.
   A description of this feature is shown in ***Figure2-2 Pair Function(Parallel execution of two pairs of processes)***.

2-1.Basic pair processing


sd Basic group processing
User Application
IMP Framework
CL que
IMP Driver
OSAL
IMP-X6 core0
IMP-X6 core1
R_IMPFW_Init()
loop
[IMP core0,IMP core1]
R_IMPFW_AttrInit(core)
pair setting of CL A
R_IMPFW_AttrSetpair(IMP core0)
R_IMPFW_AttrSetInterrupt(IMP core0)
R_IMPFW_AttrSetCl(IMP core0)
R_IMPFW_Execute(CL A:IMP core0)
enqueue the request(CL A)
return result()
pair setting of CL B
R_IMPFW_AttrSetpair(IMP core1)
R_IMPFW_AttrSetInterrupt(IMP core1)
R_IMPFW_AttrSetCl(IMP core1)
R_IMPFW_Execute(CL B:IMP core1)
enqueue the request(CL B)

Initialize data
use_core_num = 2
core_info = {IMP core0, IMP core1}

(It needs 2 times execute in this example.)
Initialize data
core_info = {core}

Pair Core setting with IMP Core1.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core1, 1 }

IMP core0 interrupt mask and group settings.
irq_group = group0
group_core_num = 2
group_core_info = { IMP, 0 }, { IMP, 1 }
interrupt_mask = { END, WUPCOVF, USIER, USINT, PBCOVF, SBO0ME, HPINT, APIPINT, MSCO }

Command List setting for IMP core0.
claddr_phys = physical address of CL A for IMP core0
clsizs = CL A data size for IMP core0
priority = PRIORITY_0

CL A does not execute in this timing because IMP core0 is pair setting to IMP core1.

Pair Core setting with IMP Core0.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core0, 0 }

IMP core1 interrupt mask and group settings.
irq_group = group0
group_core_num = 2
group_core_info = { IMP, 0 }, { IMP, 1 }
interrupt_mask = { END, WUPCOVF, USIER, USINT, PBCOVF, SBO0ME, HPINT, APIPINT, MSCO }

Command List setting for IMP core1.
claddr_phys = physical address of CL B for IMP core1
clsizs = CL B data size for IMP core1
priority = PRIORITY_0

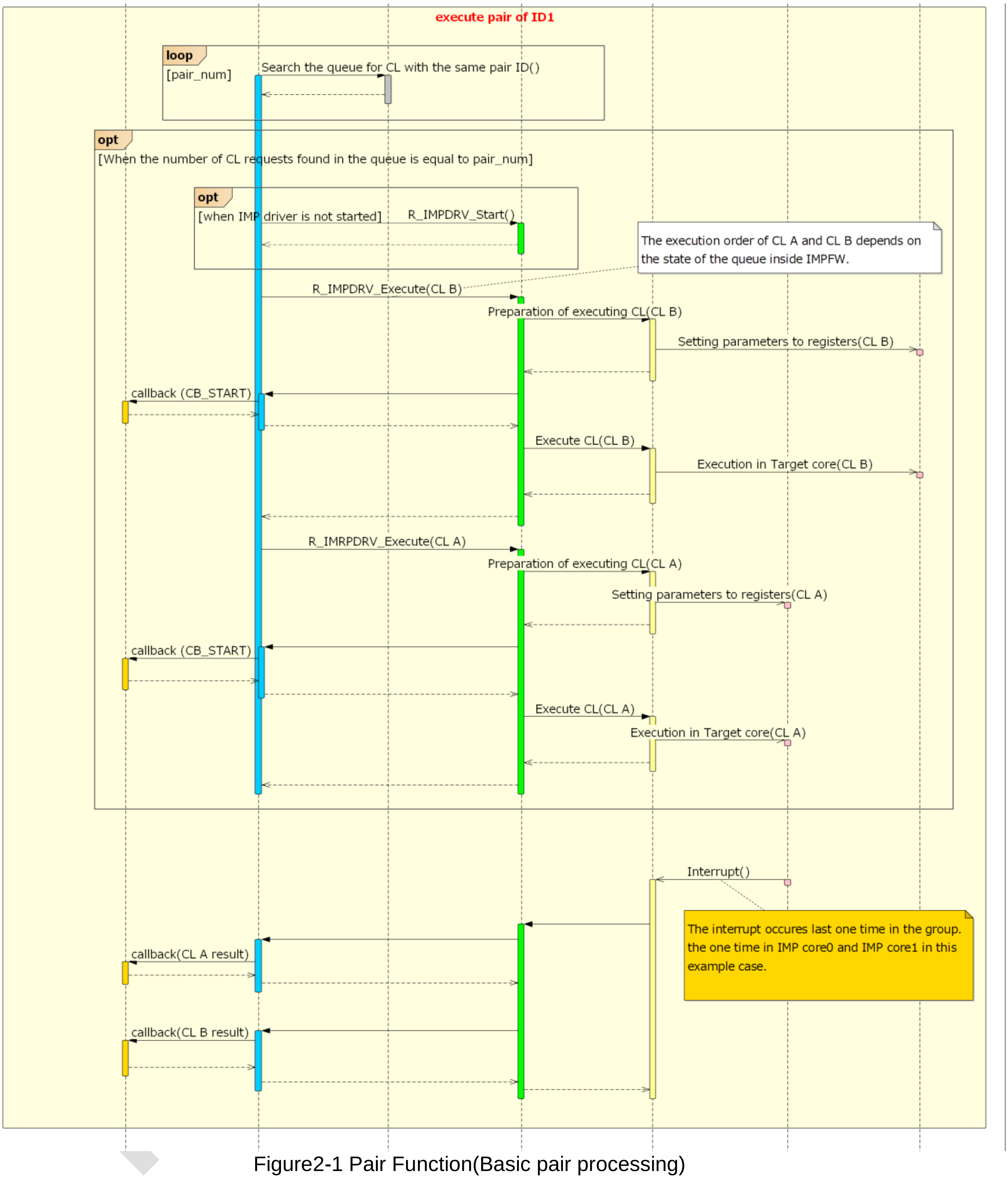
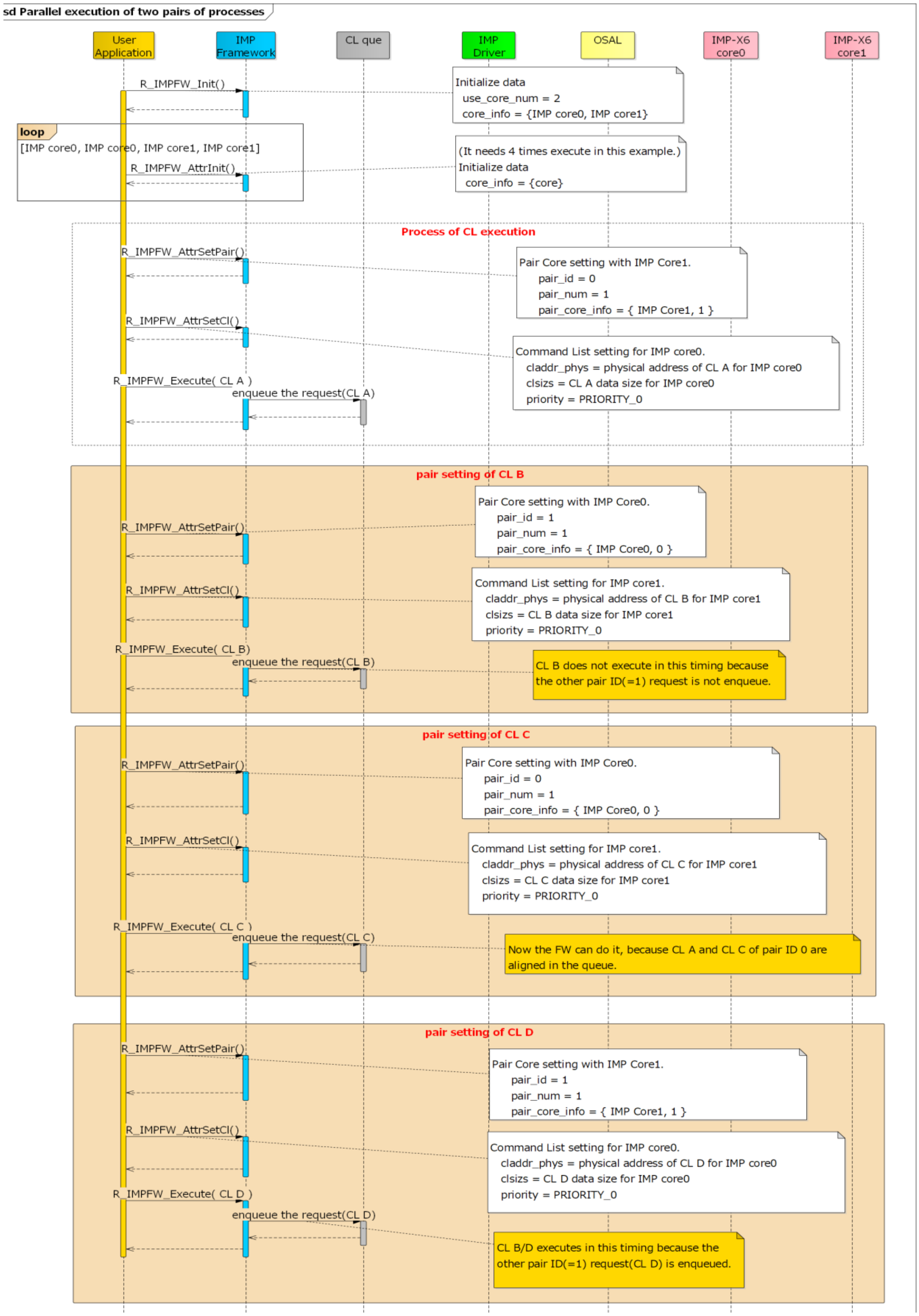

execute pair of ID1
loop
[pair_num]
Search the queue for CL with the same pair ID()
opt
[When the number of CL requests found in the queue is equal to pair_num]
opt
[when IMP driver is not started]
R_IMPDRV_Start()
The execution order of CL A and CL B depends on the state of the queue inside IMPFW.
R_IMPDRV_Execute(CL B)
Preparation of executing CL(CL B)
Setting parameters to registers(CL B)
callback (CB_START)
Execute CL(CL B)
Execution in Target core(CL B)
R_IMRPDRV_Execute(CL A)
Preparation of executing CL(CL A)
Setting parameters to registers(CL A)
callback (CB_START)
Execute CL(CL A)
Execution in Target core(CL A)
Interrupt()
The interrupt occures last one time in the group. the one time in IMP core0 and IMP core1 in this example case.
callback(CL A result)
callback(CL B result)

Figure2-1 Pair Function(Basic pair processing)

2-2.Parallel execution of two pairs of processes


sd Parallel execution of two pairs of processes
User Application
IMP Framework
CL que
IMP Driver
OSAL
IMP-X6 core0
IMP-X6 core1
R_IMPFW_Init()
Initialize data
use_core_num = 2
core_info = {IMP core0, IMP core1}
loop
[IMP core0, IMP core0, IMP core1, IMP core1]
R_IMPFW_AttrInit()
(It needs 4 times execute in this example.)
Initialize data
core_info = {core}
Process of CL execution
R_IMPFW_AttrSetPair()
Pair Core setting with IMP Core1.
pair_id = 0
pair_num = 1
pair_core_info = { IMP Core1, 1 }
R_IMPFW_AttrSetCl()
Command List setting for IMP core0.
claddr_phys = physical address of CL A for IMP core0
clsizs = CL A data size for IMP core0
priority = PRIORITY_0
R_IMPFW_Execute(CL A)
enqueue the request(CL A)
pair setting of CL B
Pair Core setting with IMP Core0.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core0, 0 }
R_IMPFW_AttrSetPair()
Command List setting for IMP core1.
claddr_phys = physical address of CL B for IMP core1
clsizs = CL B data size for IMP core1
priority = PRIORITY_0
R_IMPFW_AttrSetCl()
R_IMPFW_Execute(CL B)
enqueue the request(CL B)
CL B does not execute in this timing because the other pair ID(=1) request is not enqueue.
pair setting of CL C
R_IMPFW_AttrSetPair()
Pair Core setting with IMP Core0.
pair_id = 0
pair_num = 1
pair_core_info = { IMP Core0, 0 }
R_IMPFW_AttrSetCl()
Command List setting for IMP core1.
claddr_phys = physical address of CL C for IMP core1
clsizs = CL C data size for IMP core1
priority = PRIORITY_0
R_IMPFW_Execute(CL C)
enqueue the request(CL C)
Now the FW can do it, because CL A and CL C of pair ID 0 are aligned in the queue.
pair setting of CL D
R_IMPFW_AttrSetPair()
Pair Core setting with IMP Core1.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core1, 1 }
R_IMPFW_AttrSetCl()
Command List setting for IMP core0.
claddr_phys = physical address of CL D for IMP core0
clsizs = CL D data size for IMP core0
priority = PRIORITY_0
R_IMPFW_Execute(CL D)
enqueue the request(CL D)
CL B/D executes in this timing because the other pair ID(=1) request(CL D) is enqueued.

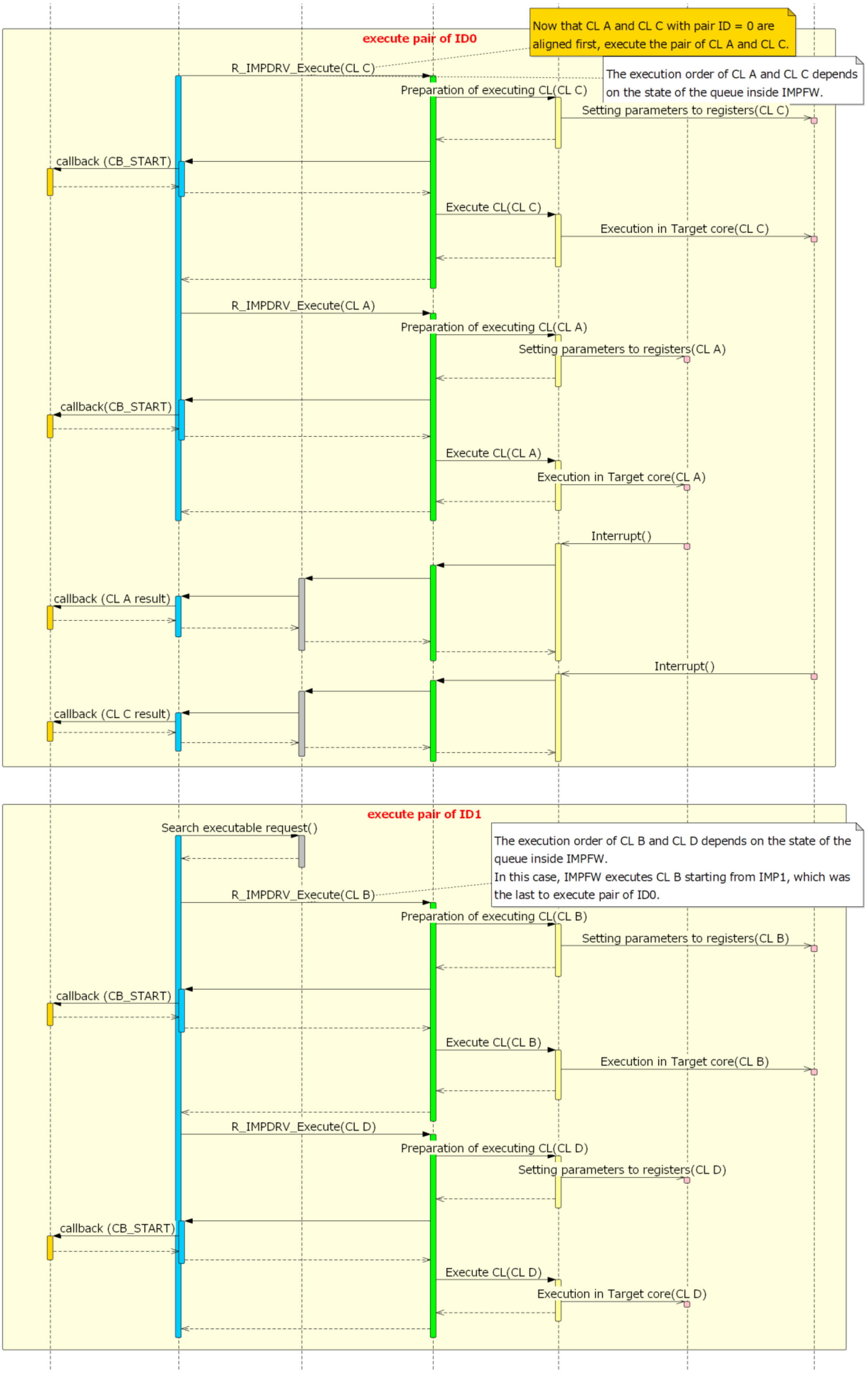

execute pair of ID0
Now that CL A and CL C with pair ID = 0 are aligned first, execute the pair of CL A and CL C.
The execution order of CL A and CL C depends on the state of the queue inside IMPFW.
R_IMPDRV_Execute(CL C)
Preparation of executing CL(CL C)
Setting parameters to registers(CL C)
callback (CB_START)
Execute CL(CL C)
Execution in Target core(CL C)
R_IMPDRV_Execute(CL A)
Preparation of executing CL(CL A)
Setting parameters to registers(CL A)
callback(CB_START)
Execute CL(CL A)
Execution in Target core(CL A)
Interrupt()
callback (CL A result)
Interrupt()
callback (CL C result)
execute pair of ID1
Search executable request()
The execution order of CL B and CL D depends on the state of the queue inside IMPFW.
In this case, IMPFW executes CL B starting from IMP1, which was the last to execute pair of ID0.
R_IMPDRV_Execute(CL B)
Preparation of executing CL(CL B)
Setting parameters to registers(CL B)
callback (CB_START)
Execute CL(CL B)
Execution in Target core(CL B)
R_IMPDRV_Execute(CL D)
Preparation of executing CL(CL D)
Setting parameters to registers(CL D)
callback (CB_START)
Execute CL(CL D)
Execution in Target core(CL D)

Figure2-2 Pair Function(Parallel execution of two pairs of processes)

3.IRQ Group function

This section describes the IRQ Group feature. Use the IRQ Group function with the following settings.

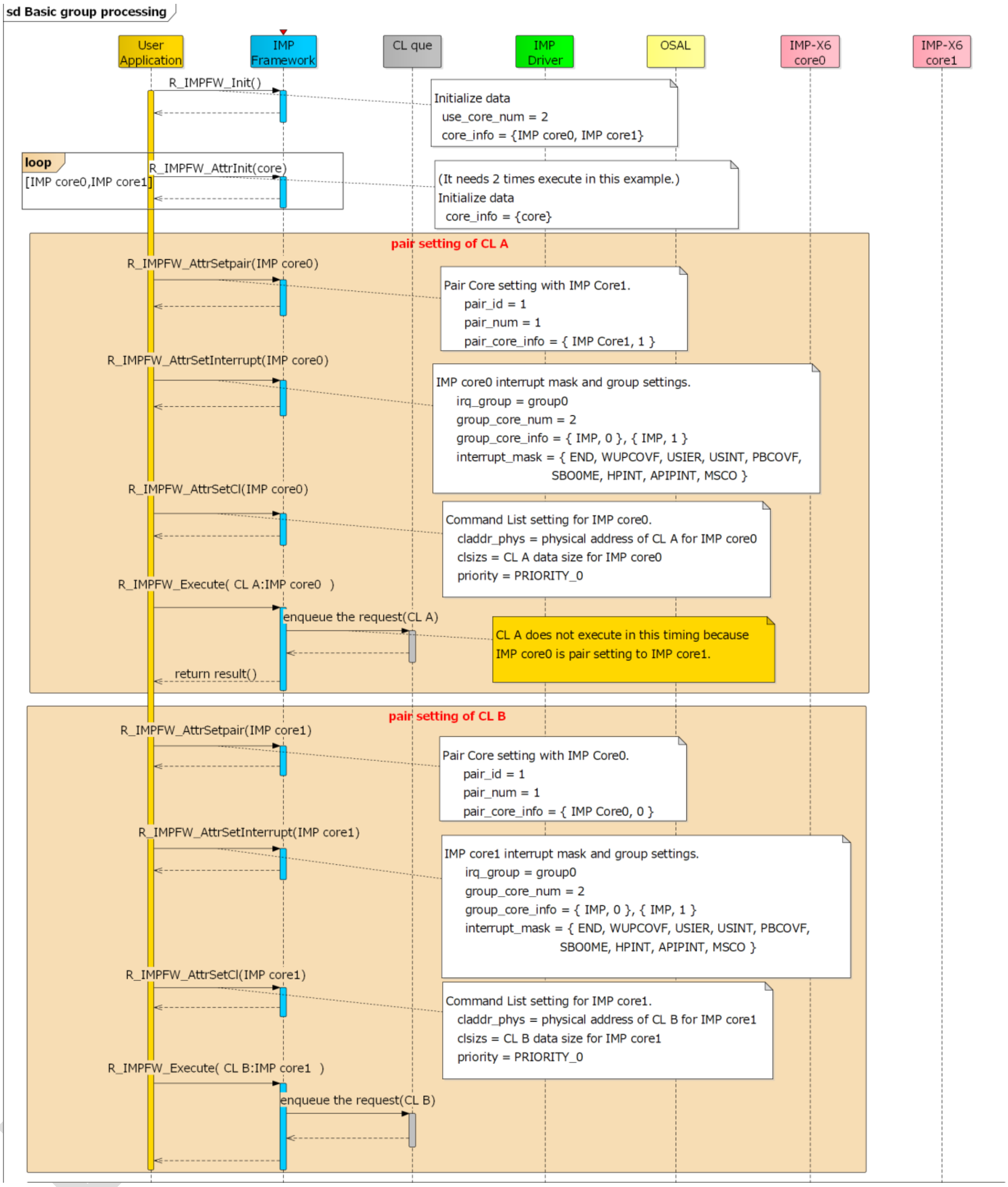
Table A-2 Parameter in ***R_IMPFW_AttrSetPair***

Pair Parameter	description
pair_core_info	In this parameter, specify the core to be paired with.
pair_num	In this parameter, specify the number of cores to be paired with.
pair_id	In this parameter, specify the ID for determining the combination of pairs.

Table A-3 Parameter in ***R_IMPFW_AttrSetInterrupt***

IRQ Group Parameter	description
irq_group	Set up the interrupt groups to register.
group_core_num	Number of cores to be grouped
group_core_info	The core information (core type, core number) to be set in the interrupt group.
interrupt_mask	Each interrupt mask information.

In IRQ Group, the interrupt occurs once in the group, and the callback to the user application is executed for the number of cores registered in the group (twice in the example).An example of setting IMP Core0 (CL A) and IMP Core1 (CL B) to IRQ Group0 is shown in ***Figure 3-1 IRQ Group Function***.


sd Basic group processing
User Application
IMP Framework
CL que
IMP Driver
OSAL
IMP-X6 core0
IMP-X6 core1
R_IMPFW_Init()
Initialize data
use_core_num = 2
core_info = {IMP core0, IMP core1}
loop
[IMP core0,IMP core1]
R_IMPFW_AttrInit(core)
(It needs 2 times execute in this example.)
Initialize data
core_info = {core}
pair setting of CL A
R_IMPFW_AttrSetpair(IMP core0)
Pair Core setting with IMP Core1.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core1, 1 }
R_IMPFW_AttrSetInterrupt(IMP core0)
IMP core0 interrupt mask and group settings.
irq_group = group0
group_core_num = 2
group_core_info = { IMP, 0 }, { IMP, 1 }
interrupt_mask = { END, WUPCOVF, USIER, USINT, PBCOVF,
SBO0ME, HPINT, APIPINT, MSCO }
R_IMPFW_AttrSetCl(IMP core0)
Command List setting for IMP core0.
claddr_phys = physical address of CL A for IMP core0
clsizs = CL A data size for IMP core0
priority = PRIORITY_0
R_IMPFW_Execute(CL A:IMP core0)
enqueue the request(CL A)
CL A does not execute in this timing because
IMP core0 is pair setting to IMP core1.
return result()
pair setting of CL B
R_IMPFW_AttrSetpair(IMP core1)
Pair Core setting with IMP Core0.
pair_id = 1
pair_num = 1
pair_core_info = { IMP Core0, 0 }
R_IMPFW_AttrSetInterrupt(IMP core1)
IMP core1 interrupt mask and group settings.
irq_group = group0
group_core_num = 2
group_core_info = { IMP, 0 }, { IMP, 1 }
interrupt_mask = { END, WUPCOVF, USIER, USINT, PBCOVF,
SBO0ME, HPINT, APIPINT, MSCO }
R_IMPFW_AttrSetCl(IMP core1)
Command List setting for IMP core1.
claddr_phys = physical address of CL B for IMP core1
clsizs = CL B data size for IMP core1
priority = PRIORITY_0
R_IMPFW_Execute(CL B:IMP core1)
enqueue the request(CL B)

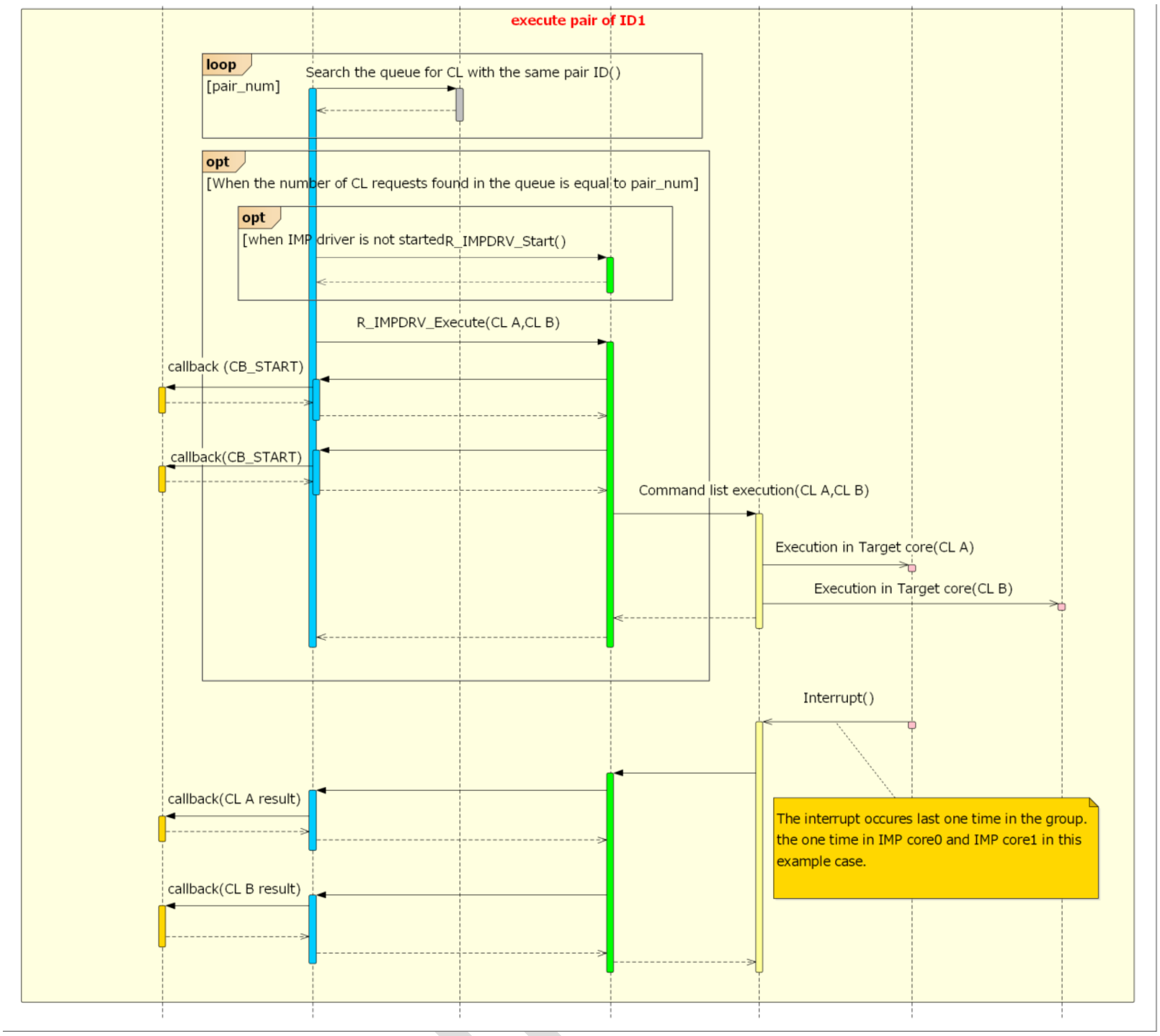

execute pair of ID1
loop
[pair_num]
Search the queue for CL with the same pair ID()
opt
[When the number of CL requests found in the queue is equal to pair_num]
opt
[when IMP driver is not started
R_IMPDRV_Start()
R_IMPDRV_Execute(CL A,CL B)
callback (CB_START)
callback(CB_START)
Command list execution(CL A,CL B)
Execution in Target core(CL A)
Execution in Target core(CL B)
Interrupt()
callback(CL A result)
callback(CL B result)
The interrupt occures last one time in the group. the one time in IMP core0 and IMP core1 in this example case.

Figure 3-1 IRQ Group Function

4.Setting CoreMap

To Synchronize the core, target modules shall be specified in WUP/SLP instruction and DPR control register. IMP-Xn can specify up to 16 target modules. IMP Framework binds the target modules which specified in the instructions or the register to the actual cores in IMP-Xn by the argument 'coremap' of ***R_IMPFW_AttrSetCoremap***. 'coremap' is an array of ***st_impfw_core_info_t***

. The index number of the array corresponds to the number of target module respectively. User shall specify the actual core which is represented by ***st_impfw_core_info_t***

in the specific element of the array. And user also shall specify all the cores which are used in the instructions of CLs. The information of coremap is written into SYNCC00 – SYNCC15 registers of each core by ***R_IMPFW_Execute***

.

For WUP/SLP instruction, each bit of 'SYNCC enable' field which has 16bits indicates the target module. The number of bit corresponds to the number of target module.

For DPR, srca_ctl or srcb_ctl field which have 4bits and takes the value from 0 to 15. The value corresponds to the number of target module.

Please refer ***Atomic Library User's Manual*** for the usage to WUP/SLP instruction and DPR control register.

A) CoreMap basic setting example

Assume number1 of coremap is set(the others are set invalid), user shall set bit[1] of 'SYNCC enable' field in WUP/SLP instruction, or set 1 to srca_id or srcb_id in DPR control register.

- Coremap value

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st_impfw_core_info_t	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-

(X : valid core type, - : invalid core type)

- 'SYNCC enable' field of WUP/SLP instruction, 'SYNCC enable' = 0x02(bit [1] is set).
- srca_ctl, srcb_ctl of DPR control register, srca_ctl or srcb_ctl = 0x01.

Note :

PSC has Execute core and Output cores. For this reason, 'PSC EXE' shall be specified for the WUP/SLP function, and each output plane(0-3) of the 'PSC OUT' shall be specified for the DPR function.

B) WUP/SLP setting example

IMP core 0, 1, 2, 3 and PSC are used, IMP core 1 and IMP core 2 use the WUP/SLP function.

Define the coremap as follows.

In this case, it is necessary to enable number1, number2 of coremap in WUP/SLP instructions in the CL.

- Coremap value

Index	***st_impfw_core_info_t***
0	IMP core 0
1	IMP core 1
2	IMP core 2
3	IMP core 3
4	PSC execute core 0
5…15	-

- ‘SYNCC enable’ field of WUP/SLP instruction, ‘SYNCC enable’ = 0x06(bit[2:1] is set).

C) DPR combined setting example

IMP core 0, 1 and PSC are used, PSC output 2 and IMP core 1 are use the DPR function.

Define the coremap as follows.

When DPR is used, WUP/SLP function can be also used in order to synchronize cores with each other.

Therefore, the CL of IMP core 1, number2 is enabled as WUP/SLP target, number1 is defined as DPR

- Coremap value

Index	***st_impfw_core_info_t***
0	PSC execute core 0
1	PSC output core 2
2	IMP core 0
3	IMP core 1
4…15	-

- ‘SYNCC enable’ field of WUP/SLP instruction, ‘SYNCC enable’ = 0x0C (bit[3:2] is set).
- srca_ctl, srcb_ctl of DPR control register, srca_ctl or srcb_ctl = 0x01.

Note :

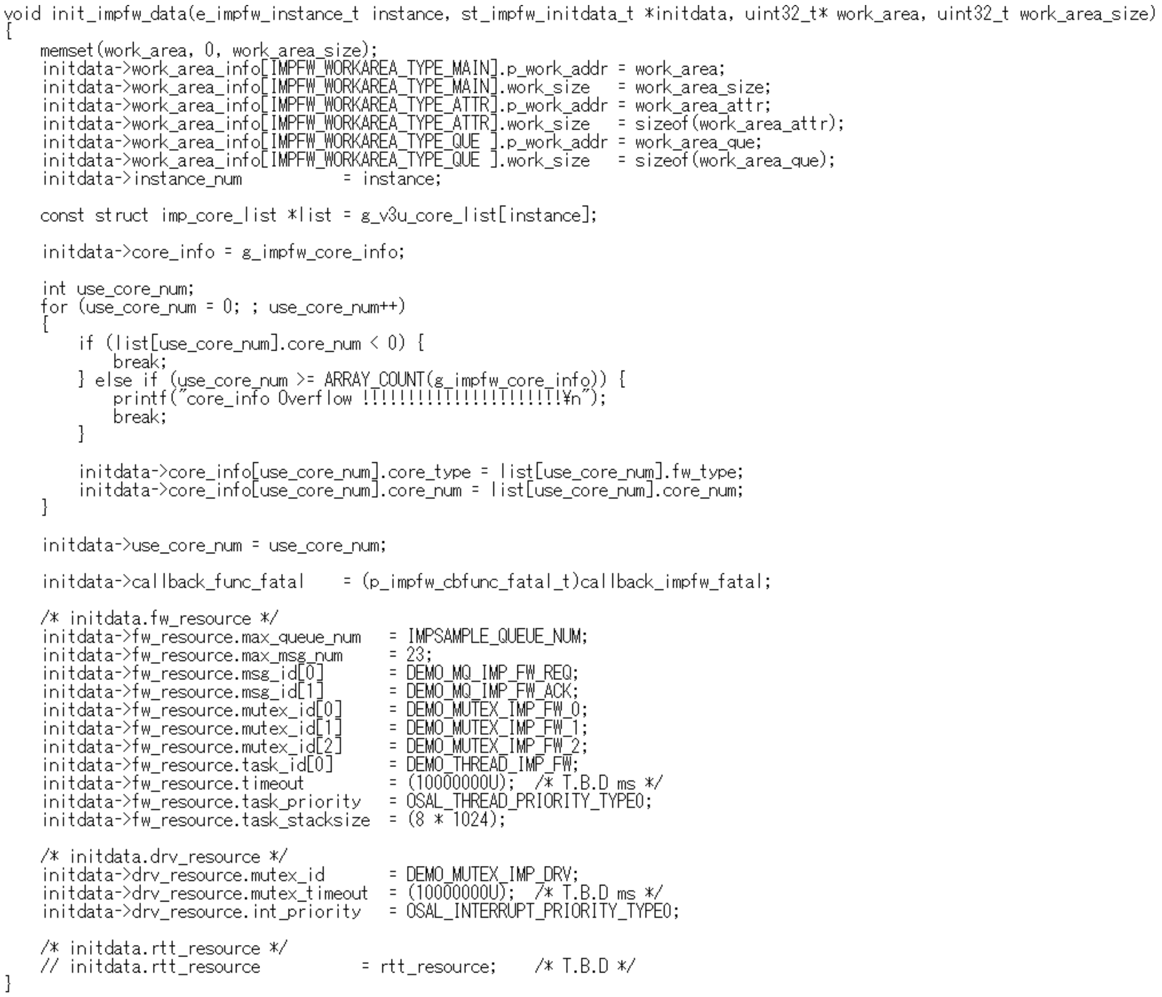
Index number0 and number1 are in the same PSC instance. So, there is 1 CL data for PSC.

5.Sample code

The code shown here is intended as a supplement when implementing the API. It is different from the sample code provided at the time of release.

5.1 Setting initialize data

The user needs to build the initialization data needed for R_IMPFW_Init in the FW. Sample code is shown in Figure 4-1.



```
void init_impfw_data(e_impfw_instance_t instance, st_impfw_initdata_t *initdata, uint32_t* work_area, uint32_t work_area_size)
{
    memset(work_area, 0, work_area_size);
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_MAIN].p_work_addr = work_area;
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_MAIN].work_size   = work_area_size;
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_ATTR].p_work_addr = work_area_attr;
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_ATTR].work_size   = sizeof(work_area_attr);
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_QUE ].p_work_addr = work_area_que;
    initdata->work_area_info[IMPFW_WORKAREA_TYPE_QUE ].work_size   = sizeof(work_area_que);
    initdata->instance_num           = instance;

    const struct imp_core_list *list = g_v3u_core_list[instance];

    initdata->core_info = g_impfw_core_info;

    int use_core_num;
    for (use_core_num = 0; ; use_core_num++)
    {
        if (list[use_core_num].core_num < 0) {
            break;
        } else if (use_core_num >= ARRAY_COUNT(g_impfw_core_info)) {
            printf("core_info Overflow !!!!!!!!!!!!!!!!!!!!!!¥n");
            break;
        }

        initdata->core_info[use_core_num].core_type = list[use_core_num].fw_type;
        initdata->core_info[use_core_num].core_num = list[use_core_num].core_num;
    }

    initdata->use_core_num = use_core_num;

    initdata->callback_func_fatal    = (p_impfw_cbfunc_fatal_t)callback_impfw_fatal;

    /* initdata.fw_resource */
    initdata->fw_resource.max_queue_num   = IMPSAMPLE_QUEUE_NUM;
    initdata->fw_resource.max_msg_num     = 23;
    initdata->fw_resource.msg_id[0]       = DEMO_MQ_IMP_FW_REQ;
    initdata->fw_resource.msg_id[1]       = DEMO_MQ_IMP_FW_ACK;
    initdata->fw_resource.mutex_id[0]     = DEMO_MUTEX_IMP_FW_0;
    initdata->fw_resource.mutex_id[1]     = DEMO_MUTEX_IMP_FW_1;
    initdata->fw_resource.mutex_id[2]     = DEMO_MUTEX_IMP_FW_2;
    initdata->fw_resource.task_id[0]      = DEMO_THREAD_IMP_FW;
    initdata->fw_resource.timeout         = (10000000U);  /* T.B.D ms */
    initdata->fw_resource.task_priority   = OSAL_THREAD_PRIORITY_TYPE0;
    initdata->fw_resource.task_stacksize  = (8 * 1024);

    /* initdata.drv_resource */
    initdata->drv_resource.mutex_id       = DEMO_MUTEX_IMP_DRV;
    initdata->drv_resource.mutex_timeout  = (10000000U);  /* T.B.D ms */
    initdata->drv_resource.int_priority   = OSAL_INTERRUPT_PRIORITY_TYPE0;

    /* initdata.rtt_resource */
    // initdata.rtt_resource            = rtt_resource;    /* T.B.D */
}
```

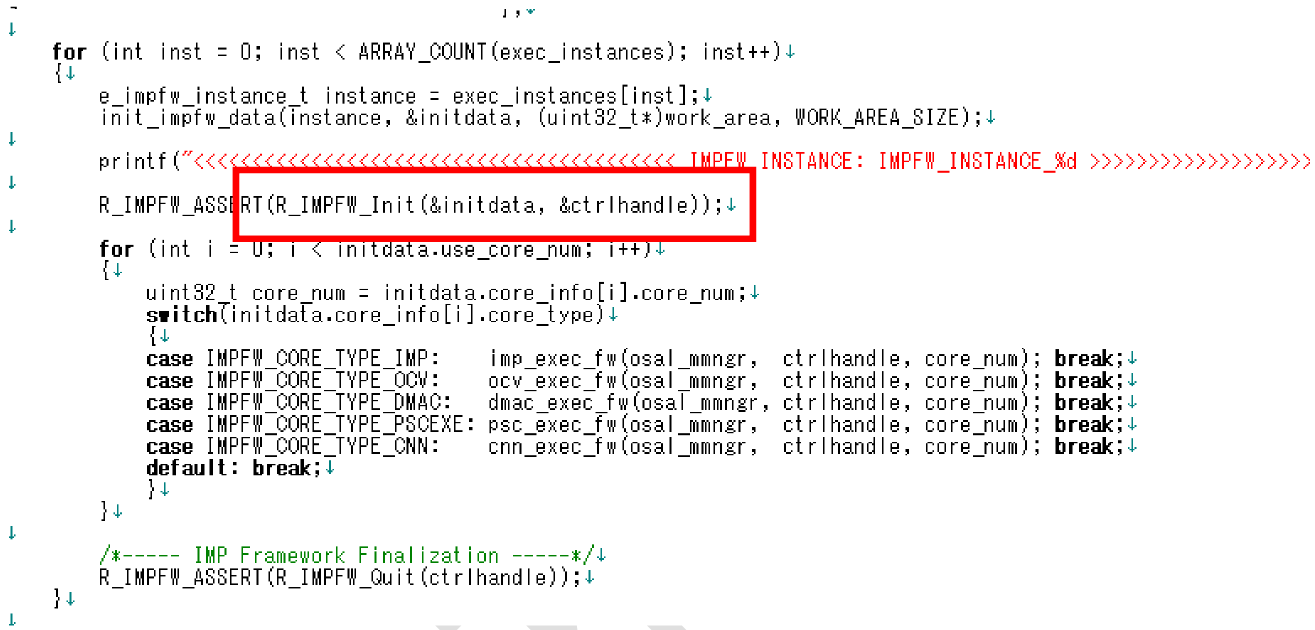
Figure 4-1 Setting initialize data

5.2 Initialize IMP Framework

The user calls R_IMPFW_Init using the created initial data to initialize the IMP Framework.

Sample code(red frame) is shown in Figure 4-2.

Variable	Description
initdata	Initialize data
ctrlhandle	Refer to ***impfw_ctrl_handle_t***



```
    for (int inst = 0; inst < ARRAY_COUNT(exec_instances); inst++)
    {
        e_impfw_instance_t instance = exec_instances[inst];
        init_impfw_data(instance, &initdata, (uint32_t*)work_area, WORK_AREA_SIZE);

        printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< IMPFW_INSTANCE: IMPFW_INSTANCE_%d >>>>>>>>>>>>>>>>>>>

        R_IMPFW_ASSERT(R_IMPFW_Init(&initdata, &ctrlhandle));

        for (int i = 0; i < initdata.use_core_num; i++)
        {
            uint32_t core_num = initdata.core_info[i].core_num;
            switch(initdata.core_info[i].core_type)
            {
            case IMPFW_CORE_TYPE_IMP:    imp_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;
            case IMPFW_CORE_TYPE_OCV:    ocv_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;
            case IMPFW_CORE_TYPE_DMAC:   dmac_exec_fw(osal_mmngr, ctrlhandle, core_num); break;
            case IMPFW_CORE_TYPE_PSCEXE: psc_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;
            case IMPFW_CORE_TYPE_CNN:    cnn_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;
            default: break;
            }
        }

        /*----- IMP Framework Finalization -----*/
        R_IMPFW_ASSERT(R_IMPFW_Quit(ctrlhandle));
    }

```

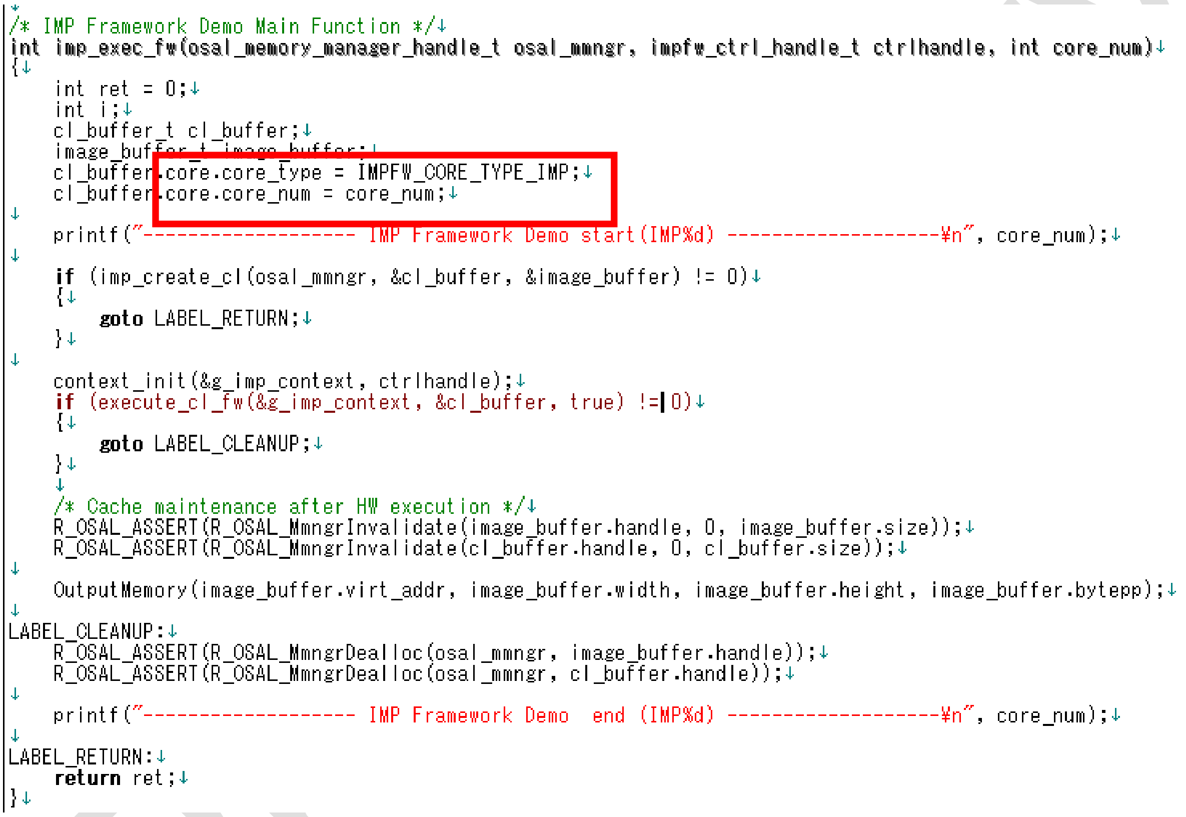
Figure 4-2 Initialize IMP Framework

5.3 Set core type for execute

The user sets the core type (core_type) and core number (core_num) to execute the command list. The sample code (red frame) when the IMP core is specified is shown in Figure 4-3.

The core type and number of cores are different for each SoC. For details, refer to ***IMP Framework Product Information: “The Core number for CL execute”***.

Abbreviation	Description
core	The core information for execute. Refer to ***st_impfw_core_info_t***



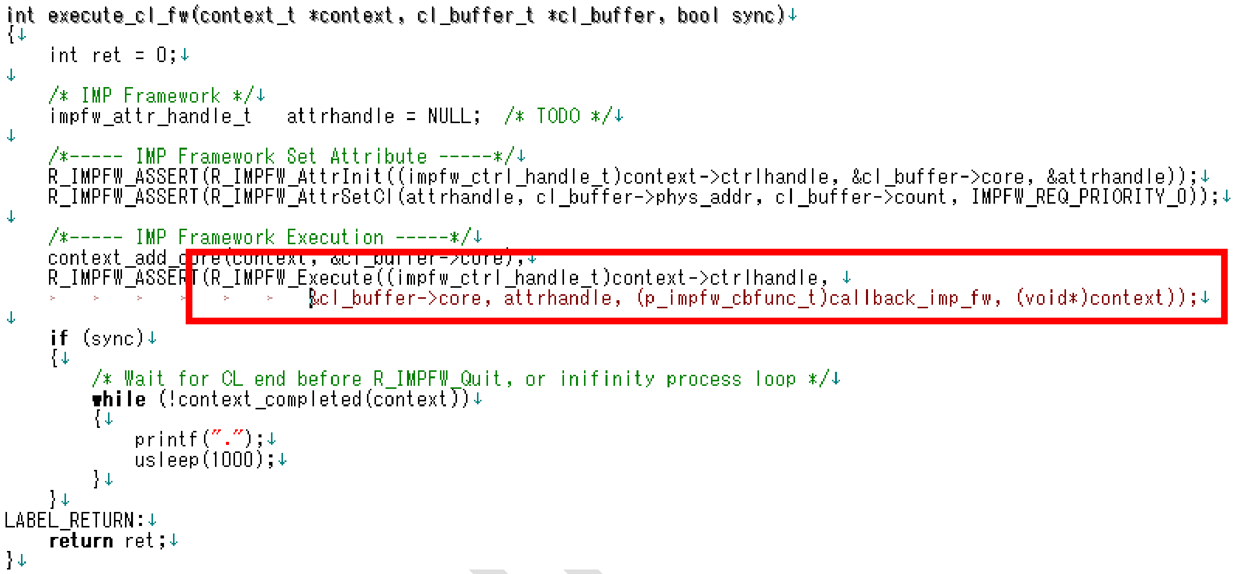
```
/* IMP Framework Demo Main Function */
int imp_exec_fw(osal_memory_manager_handle_t osal_mmngr, impfw_ctrl_handle_t ctrlhandle, int core_num)
{
    int ret = 0;
    int i;
    cl_buffer_t cl_buffer;
    image_buffer_t image_buffer;
    cl_buffer.core.core_type = IMPFW_CORE_TYPE_IMP;
    cl_buffer.core.core_num = core_num;

    printf("------------------- IMP Framework Demo start(IMP%d) ------------------¥n", core_num);

    if (imp_create_cl(osal_mmngr, &cl_buffer, &image_buffer) != 0)
    {
        goto LABEL_RETURN;
    }

    context_init(&g_imp_context, ctrlhandle);
    if (execute_cl_fw(&g_imp_context, &cl_buffer, true) != 0)
    {
        goto LABEL_CLEANUP;
    }

    /* Cache maintenance after HW execution */
    R_OSAL_ASSERT(R_OSAL_MmngrInvalidate(image_buffer.handle, 0, image_buffer.size));
    R_OSAL_ASSERT(R_OSAL_MmngrInvalidate(cl_buffer.handle, 0, cl_buffer.size));

    OutputMemory(image_buffer.virt_addr, image_buffer.width, image_buffer.height, image_buffer.bytepp);

LABEL_CLEANUP:
    R_OSAL_ASSERT(R_OSAL_MmngrDealloc(osal_mmngr, image_buffer.handle));
    R_OSAL_ASSERT(R_OSAL_MmngrDealloc(osal_mmngr, cl_buffer.handle));

    printf("------------------- IMP Framework Demo  end (IMP%d) ------------------¥n", core_num);

LABEL_RETURN:
    return ret;
}
```

Figure 4-3 Set core type for execute

5.4 Execute command List

The user executes the command list created by calling R_IMPFW_Execute. The sample code of command list execution (red frame) is shown in Figure 4-4.

Variable	Description
context->ctrlhandle	Output by ***R_IMPFW_Init***
cl_buffer->core	“core_info” specified in ***R_IMPFW_Init***
attrhandle	Output by ***R_IMPFW_AttrInit***
callback_imp_fw	Callback function pointer
context	Callback argument (if you need)



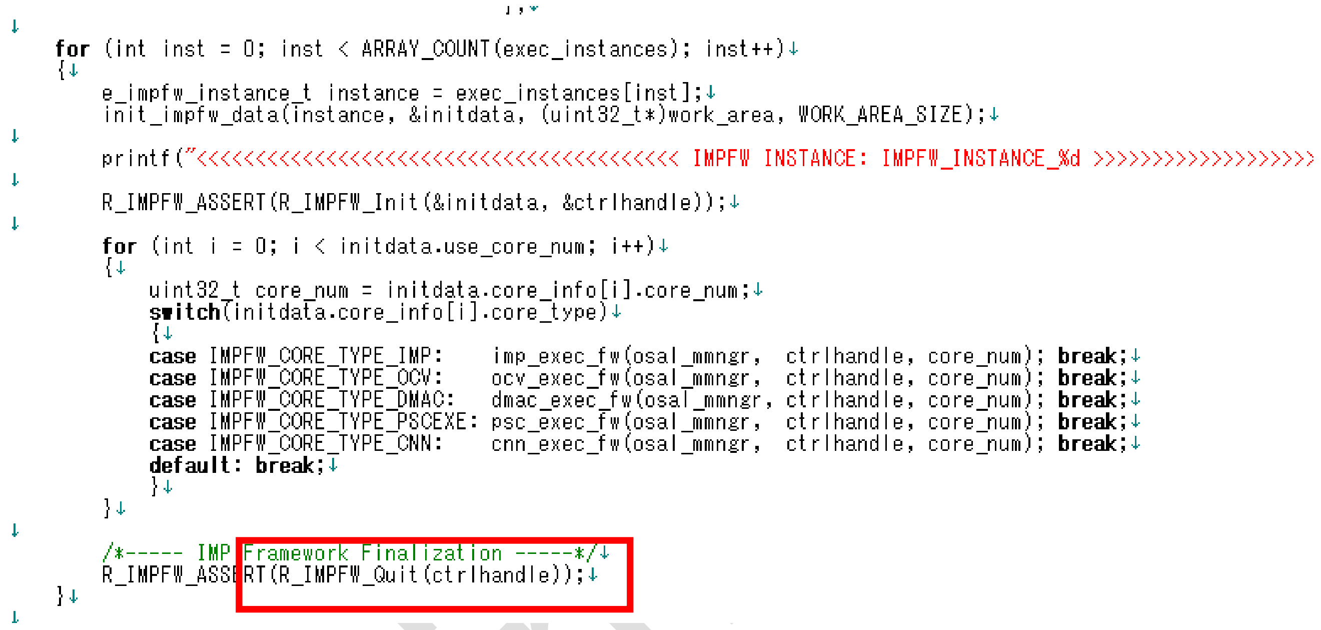
```
int execute_cl_fw(context_t *context, cl_buffer_t *cl_buffer, bool sync)
{
    int ret = 0;

    /* IMP Framework */
    impfw_attr_handle_t   attrhandle = NULL;  /* TODO */

    /*----- IMP Framework Set Attribute -----*/
    R_IMPFW_ASSERT(R_IMPFW_AttrInit((impfw_ctrl_handle_t)context->ctrlhandle, &cl_buffer->core, &attrhandle));
    R_IMPFW_ASSERT(R_IMPFW_AttrSetCl(attrhandle, cl_buffer->phys_addr, cl_buffer->count, IMPFW_REQ_PRIORITY_0));

    /*----- IMP Framework Execution -----*/
    context_add_core(context, &cl_buffer->core);
    R_IMPFW_ASSERT(R_IMPFW_Execute((impfw_ctrl_handle_t)context->ctrlhandle, 
                        &cl_buffer->core, attrhandle, (p_impfw_cbfunc_t)callback_imp_fw, (void*)context));

    if (sync)
    {
        /* Wait for CL end before R_IMPFW_Quit, or inifinity process loop */
        while (!context_completed(context))
        {
            printf(".");
            usleep(1000);
        }
    }
LABEL_RETURN:
    return ret;
}
```

Figure 4-4 Execute command list

5.5 Terminate IMP Framework

Terminate the IMP Framework. The user calls R_IMPFW_Quit to terminate the IMP Framework. The sample code (red frame) is shown in Figure 4-5.

Variable	Description
ctrlhandle	Refer to ***impfw_ctrl_handle_t***



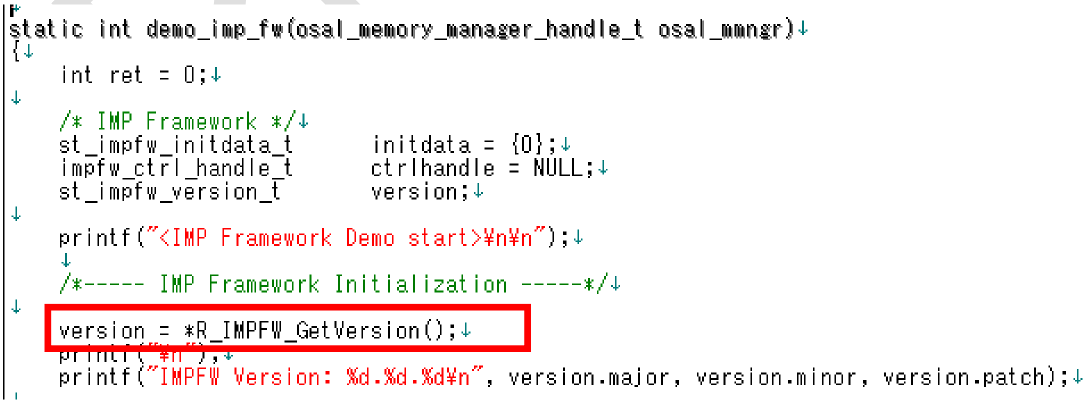
```
                                           ;↓
↓
    for (int inst = 0; inst < ARRAY_COUNT(exec_instances); inst++)↓
    {↓
        e_impfw_instance_t instance = exec_instances[inst];↓
        init_impfw_data(instance, &initdata, (uint32_t*)work_area, WORK_AREA_SIZE);↓
↓
        printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< IMPFW INSTANCE: IMPFW_INSTANCE_%d >>>>>>>>>>>>>>>>>>>
↓
        R_IMPFW_ASSERT(R_IMPFW_Init(&initdata, &ctrlhandle));↓
↓
        for (int i = 0; i < initdata.use_core_num; i++)↓
        {↓
            uint32_t core_num = initdata.core_info[i].core_num;↓
            switch(initdata.core_info[i].core_type)↓
            {↓
            case IMPFW_CORE_TYPE_IMP:    imp_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;↓
            case IMPFW_CORE_TYPE_OCV:    ocv_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;↓
            case IMPFW_CORE_TYPE_DMAC:   dmac_exec_fw(osal_mmngr, ctrlhandle, core_num); break;↓
            case IMPFW_CORE_TYPE_PSCEXE: psc_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;↓
            case IMPFW_CORE_TYPE_CNN:    cnn_exec_fw(osal_mmngr,  ctrlhandle, core_num); break;↓
            default: break;↓
            }↓
        }↓
↓
        /*----- IMP Framework Finalization -----*/↓
        R_IMPFW_ASSERT(R_IMPFW_Quit(ctrlhandle));↓
    }↓
↓
```

Figure 4-5 Terminate IMP Framework

5.6 Get IMP Framework version

Get the version of IMP Framework. The sample code (red frame) is shown in Figure 4-6.

Variable	Description
version	Refer to ***st_impfw_version_t***



```
static int demo_imp_fw(osal_memory_manager_handle_t osal_mmngr)↓
{↓
    int ret = 0;↓
↓
    /* IMP Framework */↓
    st_impfw_initdata_t        initdata = {0};↓
    impfw_ctrl_handle_t        ctrlhandle = NULL;↓
    st_impfw_version_t         version;↓
↓
    printf("<IMP Framework Demo start>¥n¥n");↓
    ↓
    /*----- IMP Framework Initialization -----*/↓
↓
    version = *R_IMPFW_GetVersion();↓
    printf("¥n");↓
    printf("IMPFW Version: %d.%d.%d¥n", version.major, version.minor, version.patch);↓
```

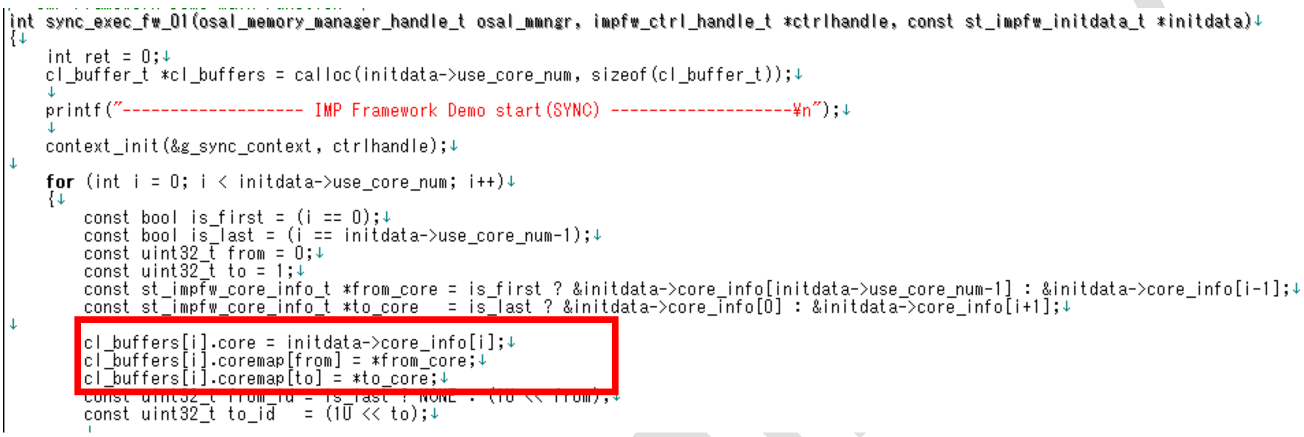
Figure 4-6 Get IMP Framework version

5.7 Setting core map for synchronize execute

A way to synchronize cores in a command list. Only cores initialized with R_IMPFW_Init can be specified for coremap, and synchronization can be achieved between cores specified with coremap.

The sample code (red frame) for setting the Core map is shown in Figure 4-7. The command list also requires a WUP / SLP to synchronize.

Variable	Description
core	“core_info” specified in ***R_IMPFW_Init***
coremap	The core to synchronize execute with above “core”



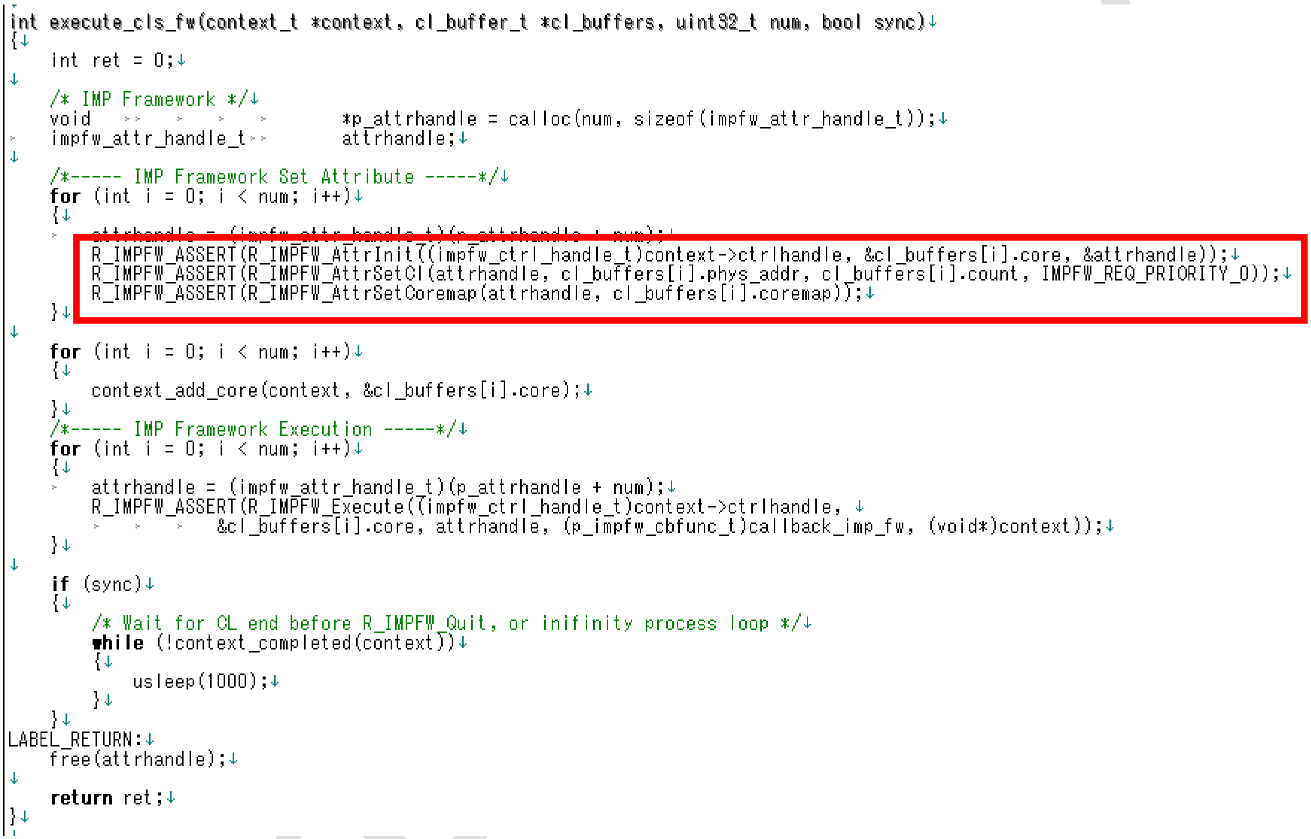
```
int sync_exec_fw_01(osal_memory_manager_handle_t osal_mmngr, impfw_ctrl_handle_t *ctrlhandle, const st_impfw_initdata_t *initdata)
{
    int ret = 0;
    cl_buffer_t *cl_buffers = calloc(initdata->use_core_num, sizeof(cl_buffer_t));

    printf("------------------ IMP Framework Demo start(SYNC) ------------------¥n");

    context_init(&g_sync_context, ctrlhandle);

    for (int i = 0; i < initdata->use_core_num; i++)
    {
        const bool is_first = (i == 0);
        const bool is_last = (i == initdata->use_core_num-1);
        const uint32_t from = 0;
        const uint32_t to = 1;
        const st_impfw_core_info_t *from_core = is_first ? &initdata->core_info[initdata->use_core_num-1] : &initdata->core_info[i-1];
        const st_impfw_core_info_t *to_core   = is_last ? &initdata->core_info[0] : &initdata->core_info[i+1];

        cl_buffers[i].core = initdata->core_info[i];
        cl_buffers[i].coremap[from] = *from_core;
        cl_buffers[i].coremap[to] = *to_core;
        const uint32_t from_id = is_last ? NONE : (1U << from);
        const uint32_t to_id   = (1U << to);
```

Figure 4-7 Setting core map information for synchronize execute

5.8 Set core map for synchronize execute to IMP Framework

The user sets the coremap information in the IMP Framework using R_IMPFW_AttrInit, R_IMPFW_AttrSetCoremap. The sample code (red frame) is shown in Figure 4-8.

Variable	Description
cl_buffers[i].core	"core_info" specified in ***R_IMPFW_Init***
cl_buffers[i]core_map	The core information by setting "***Setting core map for synchronize execute***"



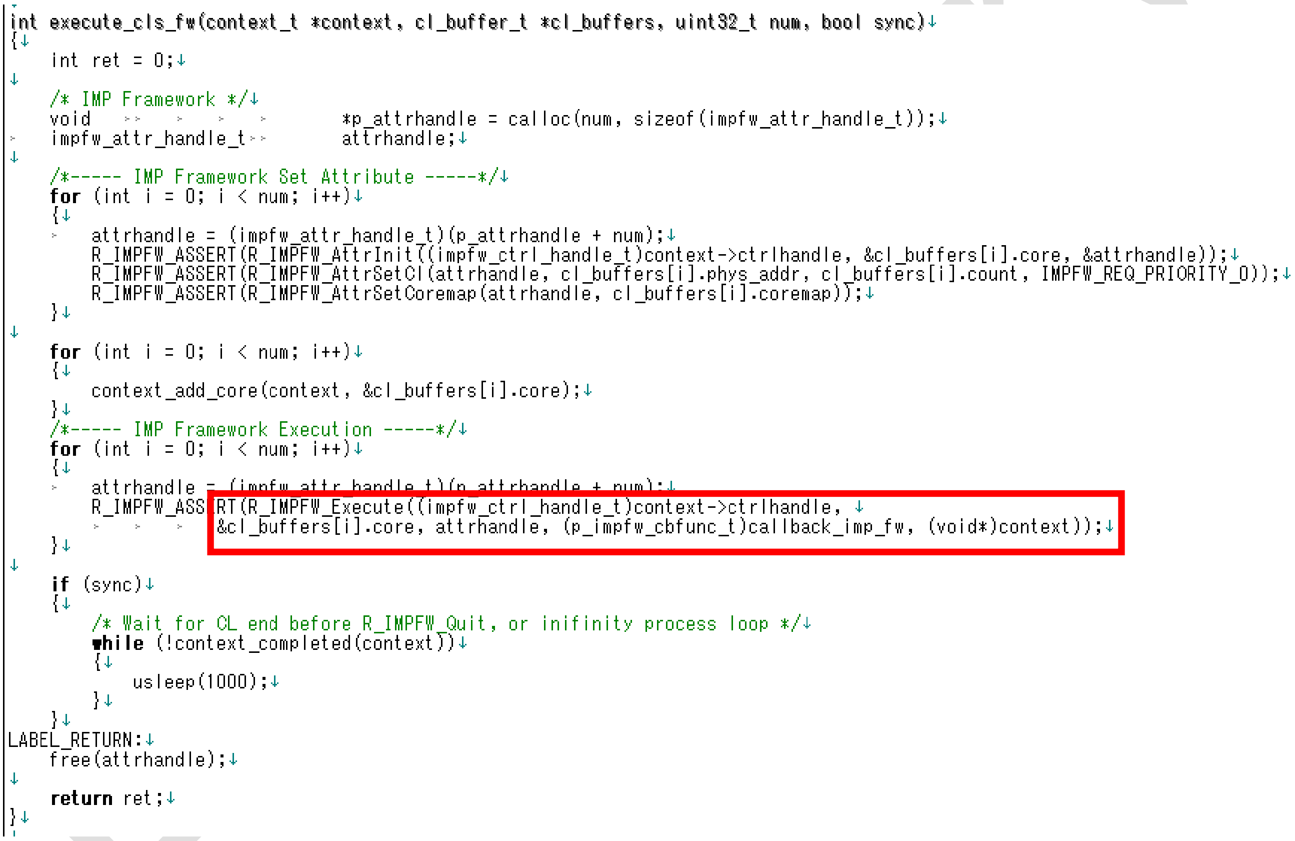
```
int execute_cls_fw(context_t *context, cl_buffer_t *cl_buffers, uint32_t num, bool sync)
{
    int ret = 0;

    /* IMP Framework */
    void                    *p_attrhandle = calloc(num, sizeof(impfw_attr_handle_t));
    impfw_attr_handle_t     attrhandle;

    /*----- IMP Framework Set Attribute -----*/
    for (int i = 0; i < num; i++)
    {
        attrhandle = (impfw_attr_handle_t)(p_attrhandle + num);
        R_IMPFW_ASSERT(R_IMPFW_AttrInit((impfw_ctrl_handle_t)context->ctrlhandle, &cl_buffers[i].core, &attrhandle));
        R_IMPFW_ASSERT(R_IMPFW_AttrSetCl(attrhandle, cl_buffers[i].phys_addr, cl_buffers[i].count, IMPFW_REQ_PRIORITY_0));
        R_IMPFW_ASSERT(R_IMPFW_AttrSetCoremap(attrhandle, cl_buffers[i].coremap));
    }

    for (int i = 0; i < num; i++)
    {
        context_add_core(context, &cl_buffers[i].core);
    }
    /*----- IMP Framework Execution -----*/
    for (int i = 0; i < num; i++)
    {
        attrhandle = (impfw_attr_handle_t)(p_attrhandle + num);
        R_IMPFW_ASSERT(R_IMPFW_Execute((impfw_ctrl_handle_t)context->ctrlhandle,
                    &cl_buffers[i].core, attrhandle, (p_impfw_cbfunc_t)callback_imp_fw, (void*)context));
    }

    if (sync)
    {
        /* Wait for CL end before R_IMPFW_Quit, or inifinity process loop */
        while (!context_completed(context))
        {
            usleep(1000);
        }
    }
LABEL_RETURN:
    free(attrhandle);

    return ret;
}
```

Figure 4-8 Set core map information to IMP Framework

5.9 Synchronize execute command list

Synchronously execute the command list using coremap. The sample code (red frame) is shown in Figure 4-9.

Variable	Description
context->ctrlhandle	Output by ***R_IMPFW_Init***
cl_buffer[i]->core	“core_info” specified in ***R_IMPFW_Init***
attrhandle	Output by ***R_IMPFW_AttrInit***
callback_imp_fw	Callback function pointer
context	Callback argument (if you need)



```
int execute_cls_fw(context_t *context, cl_buffer_t *cl_buffers, uint32_t num, bool sync)
{
    int ret = 0;

    /* IMP Framework */
    void                            *p_attrhandle = calloc(num, sizeof(impfw_attr_handle_t));
    impfw_attr_handle_t             attrhandle;

    /*----- IMP Framework Set Attribute -----*/
    for (int i = 0; i < num; i++)
    {
        attrhandle = (impfw_attr_handle_t)(p_attrhandle + num);
        R_IMPFW_ASSERT(R_IMPFW_AttrInit((impfw_ctrl_handle_t)context->ctrlhandle, &cl_buffers[i].core, &attrhandle));
        R_IMPFW_ASSERT(R_IMPFW_AttrSetCl(attrhandle, cl_buffers[i].phys_addr, cl_buffers[i].count, IMPFW_REQ_PRIORITY_0));
        R_IMPFW_ASSERT(R_IMPFW_AttrSetCoremap(attrhandle, cl_buffers[i].coremap));
    }

    for (int i = 0; i < num; i++)
    {
        context_add_core(context, &cl_buffers[i].core);
    }
    /*----- IMP Framework Execution -----*/
    for (int i = 0; i < num; i++)
    {
        attrhandle = (impfw_attr_handle_t)(p_attrhandle + num);
        R_IMPFW_ASSERT(R_IMPFW_Execute((impfw_ctrl_handle_t)context->ctrlhandle,
                    &cl_buffers[i].core, attrhandle, (p_impfw_cbfunc_t)callback_imp_fw, (void*)context));
    }

    if (sync)
    {
        /* Wait for CL end before R_IMPFW_Quit, or inifinity process loop */
        while (!context_completed(context))
        {
            usleep(1000);
        }
    }
LABEL_RETURN:
    free(attrhandle);

    return ret;
}
```

Figure 4-9 synchronize execute command list

Revision History		IMP Framework User’s Manual	
Rev.	Date	Description	
		Page	Summary
0.10E	Dec 20, 2020	-	The following is the change history from User's Manual of IMPFW for xOS1.
		1 - 3	Fixed contents for V3U.
		4	Fixed contents for xOS2.
		5.2.7 – 5.2.12	Fixed contents for xOS2. - Added the new APIs for setting CL execution information.
0.11E	Apr 15, 2021	11	Change Figure3-2 of 3.2,
		61	Figure2-2 of Appendix
0.12E	May 20, 2021	4-8	Fixed contents for V3M/V3H.
		24-31	Add detail description and valid value table
		33-56	Modify variable name
		63-70	Add Appendix for implementation example
0.13E	Jun 10, 2021	11	Added description about CL execution processing
		12	Change basic execution flow
		13	Add parallel execution flow
		16	Table 3-1 No.2 Change “Working memory area” to “Management memory area”. Delete “or more” of “IMPFW_WORKAREA_TYPE_MAIN : 13KB or more”.
		21-22	4.3.2 Add “IMPFW_CB_USINT”, and change member position. Add Table 4-5 for user behavior.
		32	Table4-24 Added the description when 0 is set to "group_core_num". Table4-25 Added the description when NULL is set to "group_core_info".
		33	4.4.9 Delete “int_mask, trap_mask, ier_mask”
		51	5.2.9 Description Add description for pair cancel setting. Add description for pair ID reuse.
		54	5.2.10 Description Add description for coremap cancel setting.
		56	5.2.11 Description Add description for IRQ Group setting example.
		57	5.3.1 Change return type to “int32_t”
		63-66	Figure2-1 , Figure2-2 Add R_IMPFW_Init, R_IMPFW_AttrInit, R_IMPFW_AttrSetpair and R_IMPFW_AttrSetCl to sequence.
		67	Add 3. IRQ Group function
0.14E	Jun 15, 2021	70, 71	Appendix 4. Setting CoreMap Correction of setting method
		12	3.2.1 Basic Execution Figure 3-2 Function Flow diagram was modified.
		13	3.2.2 Parallel Execution Figure 3-3 Function Flow diagram of parallel execute was modified.

		16	3.4.1 User side prepare data IMPF_V3M_X20QNX_D_ProductInfo_03_usage.docx 3. Added Abbreviation.
		51	5.2.9 R_IMPFW_AttrSetPair Fixed Rerfer spelling to be correct.
		63, 64	Appendix 2-1. Basic pair processing Figure 2-1 Pair Function (Basic pair processing) was modified.
		65, 66	Appendix 2-2. Parallel execution of two pairs of processes Figure 2-2 Pair Function (Parallel execution of two pairs of processes) was modified.
		42	5.2.4 R_IMPFW_Resume "Not implement" is written after the API name.
		44	5.2.5 R_IMPFW_SetPmPolicy "Not implement" is written after the API name.
		55	5.2.11 R_IMPFW_AttrSetInterrupt "Not implement" is written after the API name.
		27	4.4.1 st_impfw_core_info_t Added the chapter number of IMP Framework Product Information.
		23	4.3.3 e_impfw_core_type_t Same description as HWM.
		7	1.2 References Removed reference to Safety Application Note.
		7	1.3 List of Terms Corrected the description of Runtime Test and Safety Mechanism.
		14	3.3.4 IMP DRIVER ERROR The coping method is described.
		21, 22	4.3.2 e_impfw_callback_reason_t Table 4-4 Removed (T.B.D.) from Enumerator of e_impfw_callback_reason_t. Table 4-5 Removed (T.B.D.) from User behavior of e_impfw_callback_reason_t.
		26	4.3.8 e_impfw_fatalcode_t Deleted (T.B.D.).
		29	4.4.3 st_impfw_initdata_t Fixed (T.B.D.) to "Not implement".
		30	4.4.4 st_impfw_fw_resource_t The description of the maximum value is described in the Description of max_queue_num. The description of the maximum value is described in the Description of max_msg_num.
		59	5.3.2 p_impfw_cbfunc_fatal_t Fixed (T.B.D.) to Not implement. The coping method is described in Description.
		39	5.2.2 R_IMPFW_Execute Described the interrupt generated by Default in Description. Added a limit to *p_core_info in <input parameters>.
0.15E	Jun 30, 2021	21, 22	4.3.2 e_impfw_callback_reason_t Added IMPFW_CB_UDIVSBRK and IMPFW_CB_UDIPSBRK. Changed the value of enum.

0.16E		47	5.2.7 R_IMPFW_AttrInit Removed 'const' from the type of p_attrhandle.
0.17E	Jul 2, 2021	42	5.2.4 R_IMPFW_Resume Removed IMPFW_EC_NG_DRVERROR from Return Codes.
0.18E	Jul 5, 2021	8	2.1 Summary Specification Change of description content for the 'Summary Specification'.
0.19E	Jul 7, 2021	22	4.2 Definition Values Changed IMPFW_VERSION_MINOR.
		30	4.3.8 e_impfw_fatalcode_t Fixed fatal error type.
		35	4.4.4 st_impfw_fw_resource_t Changed the definition of the following members to the definition of OSAL. msg_id, mutex_id, task_id, timeout, task_priority
		36	4.4.5 st_impfw_drv_resource_t Changed the definition of the following members to the definition of OSAL. mutex_id, mutex_timeout, int_priority
		44	5.2.2 R_IMPFW_Execute Removed const from parameter "impfw_ctrl_handle_t handle".
		49	5.2.4 R_IMPFW_Resume Removed const from parameter "impfw_ctrl_handle_t handle".
		51	5.2.5 R_IMPFW_SetPmPolicy (Not implement) Added const to "e_impfw_pmpolicy_t policy" of Function Prototypes.
		80	Appendix 5.1 Setting initialize data Figure 4-1 Setting initialize data was modified.
0.20E	Jul 12, 2021	7	1.2 References Updated revision number of Product information.
		8	2.1 Summary Specification Fixed the description of Execution function for Command list.
		9	2.1 Summary Specification Added the description to Core map settings.
		64, 65	5.3.1 p_impfw_cbfunc_t Changed the name of arguments. Added 'const' to the pointer of p_core_info.
		42, 44, 49, 51, 54, 56, 58, 60, 62, 64	In the following sections, removed unnecessary indentions in Function Prototypes. 5.2.1 R_IMPFW_Init 5.2.2 R_IMPFW_Execute 5.2.4 R_IMPFW_Resume 5.2.5 R_IMPFW_SetPmPolicy 5.2.7 R_IMPFW_AttrInit 5.2.8 R_IMPFW_AttrSetCl 5.2.9 R_IMPFW_AttrSetPair 5.2.10 R_IMPFW_AttrSetCoremap 5.2.11 R_IMPFW_AttrSetInterrupt 5.3.1 p_impfw_cbfunc_t

CONFIDENTIAL

R-Car IMP Framework
User's Manual

Publication Date: Rev.0.20E Jul 12, 2021

Published by: Renesas Electronics Corporation

IMP Framework

Renesas Electronics Corporation