User's Manual

RENESAS

R-Car Series, 3rd Generation
IMR Device Driver for INTEGRITY ®

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.0.51 Apr 2017

Trademark

- Green Hills, the Green Hills logo, INTEGRITY, MULTI, DoubleCheck, EventAnalyzer, Integrate,SuperTrace, ResourceAnalyzer, CodeFactor, INTEGRITY MULTIvisor, GMART, GSTART, G-Cover,PathAnalyzer, GHNet, TimeMachine, µ-velOSity, Padded Cell, TotalDeveloper, and Optimizing Compiler are trademarks or registered trademarks of Green Hills Software in the US and/or internationally.
- Trademarks and trademark symbols (® or ™) are omitted in the text of this manual.

How to Use This Manual

1. Purpose and Target Readers

This manual is designed to provide the user with an understanding the functions of IMR Device Driver for INTEGRITY. This manual is written for engineers who use IMR.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

The following documents apply to the R-Car 3rd Generation. Make sure to refer to the latest versions of these documents. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

Document Type	Description	Document Title	Document No.
User's manual for Hardware	Hardware specifications (pin assignments, memory maps, peripheral function specifications, electrical characteristics, timing charts) and operation description Note: Refer to the application notes for details on using peripheral functions.	R-Car 3rd Generation User's Manual: Hardware	

2. Notation of Numbers and Symbols

This manual uses the following notation.

Binary 0bXXXXXXXX (X=0 or 1)
Decimal XXX (X=0-9)
Hex 0xXXXXXXXX (X=0-9,A-F)

3. List of Abbreviations and Acronyms

Abbreviation	Full Form
BSP	Board Support Package
bpp	bytes per pixel
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
I/O	Input/Output
LSB	Least Significant Bit
MSB	Most Significant Bit
NC	Non-Connect

Table of Contents

1. Overview

1.1 Specifications

IMR device driver (hereafter called “this software”) is a software to initialize IMR-LX4, execute the rendering, and operate the Display List. This software runs on Green Hills Software’s Real-Time OS INTEGRITY®. See INTEGRITY-related documents for INTEGRITY® specifications, etc.

Table 1.1 shows the image data specifications corresponding to this software.

Table 1.1 Image Data Specifications

Item	Description
Source Image Data Format	YUV422 Brightness: Y 8 bpp Color difference: UV 8 bpp
Destination Image Data Format	YUV422 Brightness: Y 8 bpp Color difference: UV 8 bpp
Combination of Source and Destination	1. Source: Y/UV separate format, Destination: Y/UV separate format 2. Source: Interleaved format, Destination: Y/UV separate format 3. Source: Interleaved format, Destination: Interleaved format
Drawing Area	Source coordinates (u, v): 0 ≤ u ≤ 2,047, 0 ≤ v ≤ 2,047 Destination coordinates (X, Y): 0 ≤ X ≤ 2,047, 0 ≤ Y ≤ 2,047

2. Software Description

2.1 Operation Overview

2.1.1 Software Block Diagram

Figure 2.1 shows the block diagram of this software. This software uses Memory Manager (hereafter called "MMGR") module.

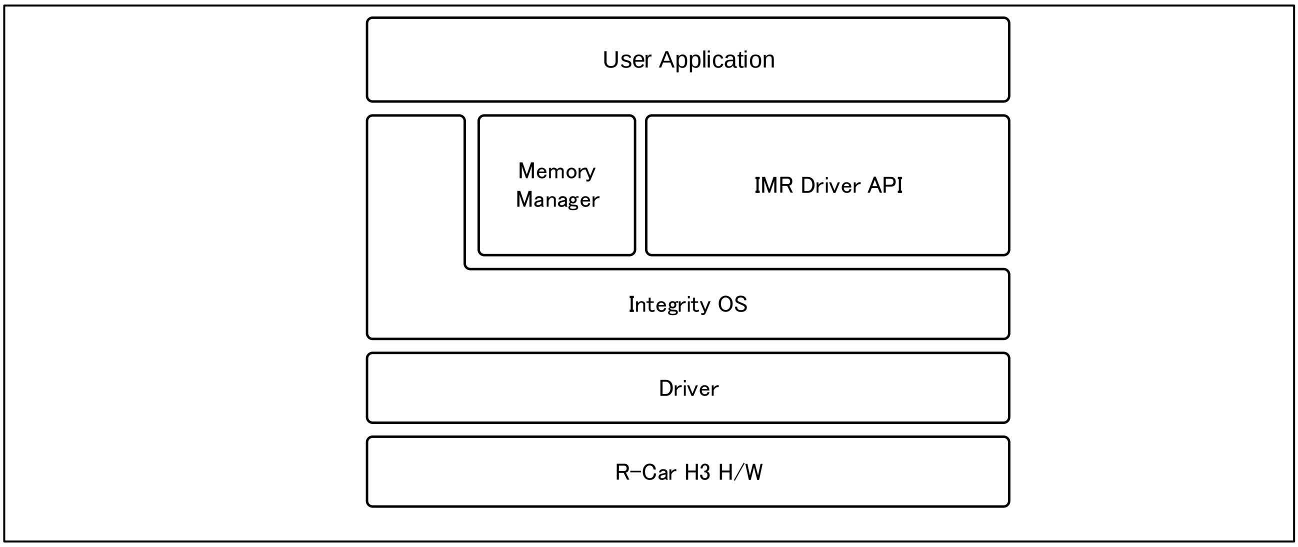


Figure 2.1 Software Block Diagram

2.1.2 Data Flow

Figure 2.2 shows the rendering processing data flow using this software.

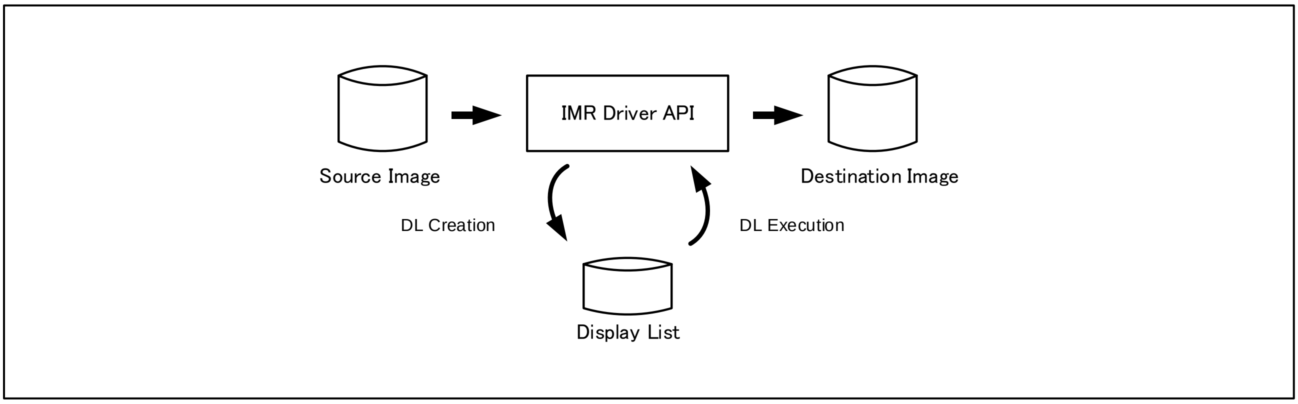


Figure 2.2 Data Flow

2.1.3 IMR-LX4 Initialization

Power supply and clock supply to the IMR-LX4 module are carried out within OS at boot time.

2.1.4 Driver Software States

This software manages the execution states of APIs using five driver software states for each IMR-LX4 channel. Table 2.1 lists the driver software states. If an API that is not corresponding to the sate is executed, the API returns a state error (E_OBJ). Figure 2.3 shows the state transition diagram of this software.

Table 2.1 Driver Software States

State	Description
Uninitialization	A state at reset release. Only initialization function (R_IMR_Init) is valid.
Stop	A state in which this software is initialized.
Close	A state in which this software is started.
Open	A state in which this software is opened. APIs for the DL execution can be executed from this state.
DL Running	A state in which the rendering is being executed.

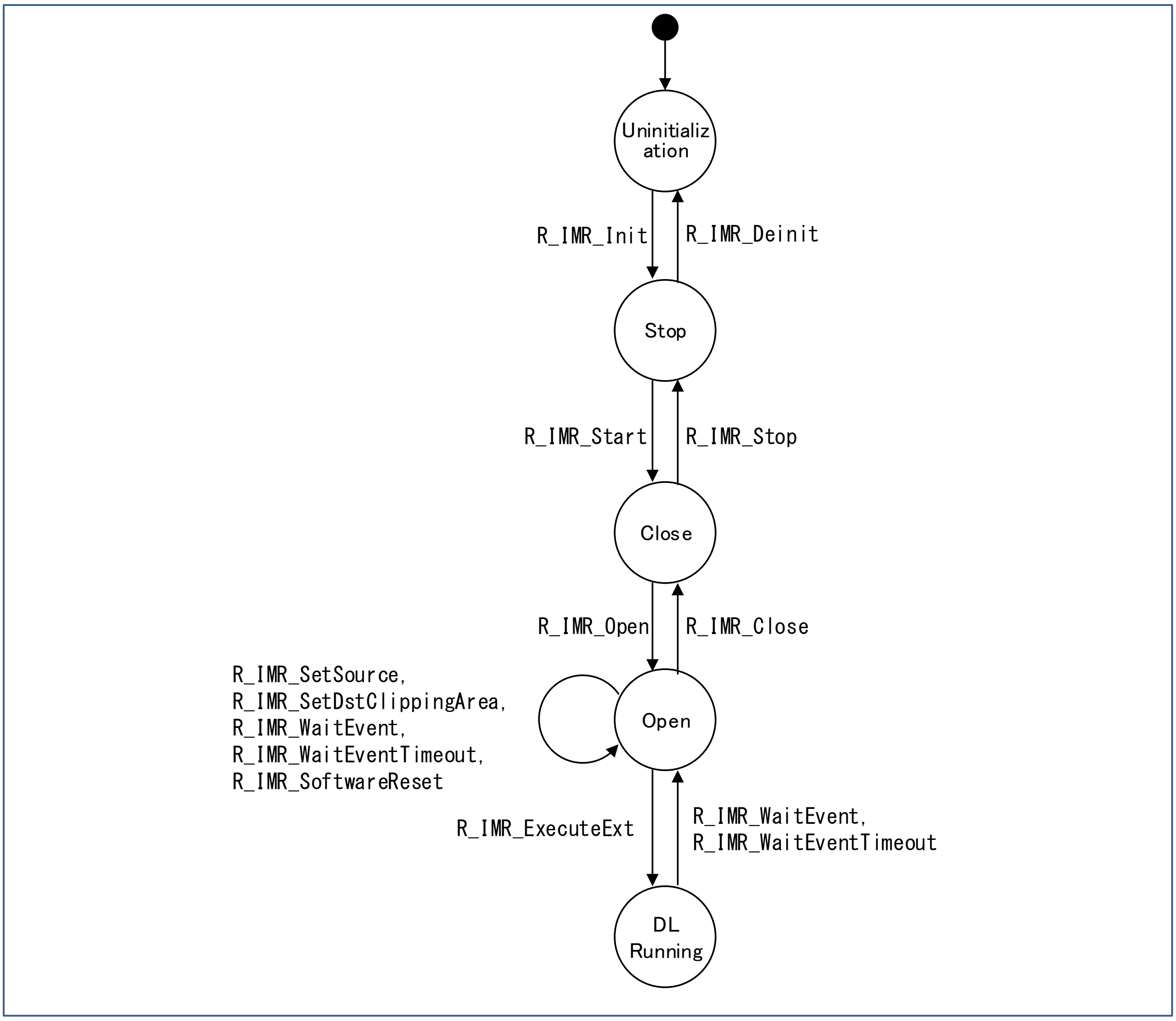
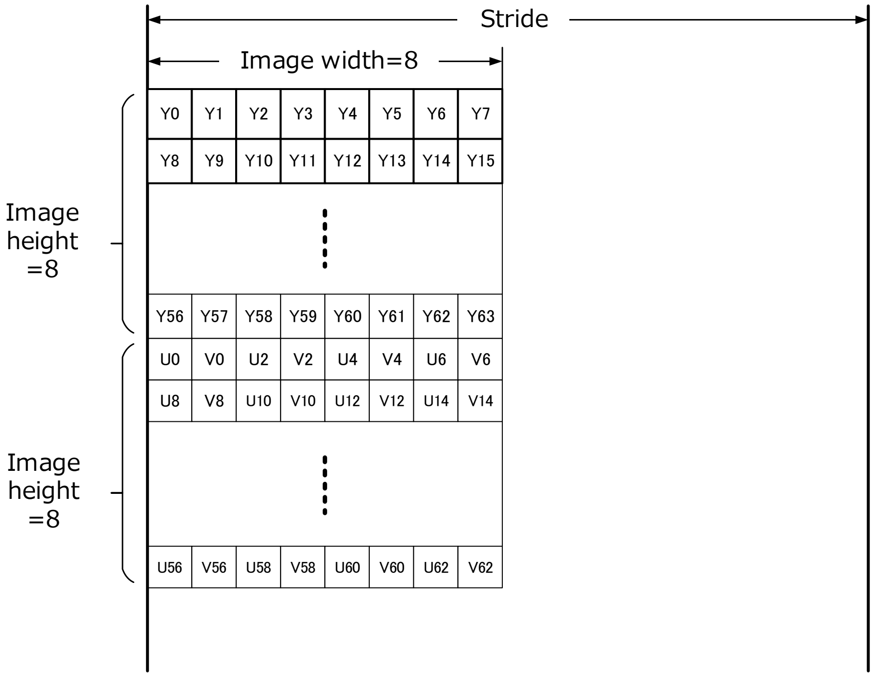


Figure 2.3 State Transition Diagram

2.1.5 Memory Region for Images

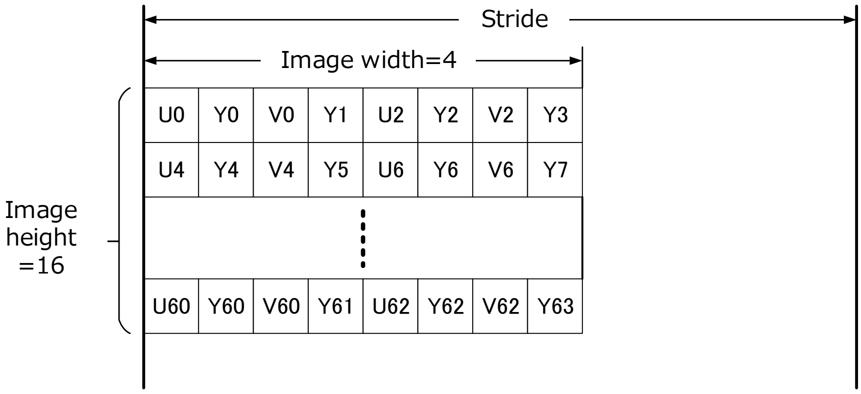
Figure 2.4 shows the memory imagery of the Y/UV separation format, and Figure 2.5 shows the memory imagery of the interleaved format.



Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
			⋮				
Y56	Y57	Y58	Y59	Y60	Y61	Y62	Y63
U0	V0	U2	V2	U4	V4	U6	V6
U8	V8	U10	V10	U12	V12	U14	V14
			⋮				
U56	V56	U58	V58	U60	V60	U62	V62

Example of image width: 8 pixels and image height: 8 lines

Figure 2.4 Memory Imagery of Y/UV Separation Format



U0	Y0	V0	Y1	U2	Y2	V2	Y3
U4	Y4	V4	Y5	U6	Y6	V6	Y7
			⋮				
U60	Y60	V60	Y61	U62	Y62	V62	Y63

Example of image width: 4 pixels, image height: 16 lines, and YCFORM=1

Figure 2.5 Memory Imagery of Interleaved Format

2.1.6 Display List

Rendering by IMR is performed by specifying a Display List (hereafter called "DL"), an array of data in 4-byte units, where the operation codes such as IMR register operation and conversion coordinates designation are described, to IMR. In this document, this 4-byte data is described as a DL element. Also, one or more DL elements combining the operation code and data required for that operation code is described as a DL instruction.

The DL can be created by the user application using the DL operation APIs of this software. Also, DL of a fixed table created in advance can be used by copying it into DL memory area.

2.2 List of Structures/Unions

This section explains structures/unions to be used by this software.

2.2.1 r_imr_2dpos_t

The r_imr_2dpos_t type is a type that defines two-dimentional coordinates. Figure 2.6 shows the r_imr_2dpos_t type.

```
#define r_imr_2dpos_t T_IMRDRV_2DPOS
typedef struct
{
   int32_t X;          /* X coordinate */
   int32_t Y;          /* Y coordinate */
} T_IMRDRV_2DPOS;
```

Figure 2.6 r_imr_2dpos_t Type

2.2.2 r_imr_rect_t

The r_imr_rect_t type is a type that defines a rectangle. Figure 2.7 shows the r_imr_rect_t type.

```
#define r_imr_rect_t T_IMRDRV_RECT
typedef struct
{
   int32_t X;         /* X coordinate of the upper left of the rectangle */
   int32_t Y;         /* Y coordinate of the upper left of the rectangle */
   uint32_t Width;    /* Rectangle width (pixel) */
   uint32_t Height;   /* Rectangle height (line) */
} T_IMRDRV_RECT;
```

Figure 2.7 r_imr_rect_t Type

2.2.3 r_imr_source_t

The r_imr_source_t type is a type that defines the height and width of a source image to be rendered. Figure 2.8 shows the r_imr_source_t type.

```
#define r_imr_source_t T_IMRDRV_SOURCE
typedef struct
{
   uint32_t start_line;       /* Beginning line (unused) */
   uint32_t end_line;         /* End line (unused) */
   uint32_t mesh_size;        /* Mesh size (unused) */
   uint32_t source_width;     /* Source image width (pixel) */
   uint32_t source_height;    /* Source image height (line) */
} T_IMRDRV_SOURCE;
```

Figure 2.8 r_imr_source_t Type

2.2.4 r_imr_dl_t

The r_imr_dl_t type is a type that defines the address and the size of ID storage area. Set the address of the DL storage area obtained by R_MMGR_AllocMemory () to PhysAddr and VmrStartAddr Figure 2.9 shows the r_imr_dl_t type.

```
#define r_imr_dl_t T_IMR_DL
typedef struct
{
   uint32_t Size;         /* Size of DL storage area (bytes) */
   uint32_t Pos;          /* Member to be used in the driver */
   uint32_t PhysAddr;     /* Physical address of DL storage area */
   Address  VmrStartAddr; /* Vritual address of storage area*/
} T_IMR_DL;
```

Figure 2.9 r_imr_dl_t Type

2.2.5 r_imr_data_t

The r_imr_data_t type is a type that defines the start address and format of the picture. Set the address of the DL storage area obtained by R_MMGR_AllocMemory () to PhysAddr and VmrStartAddr. Figure 2.10 shows the r_imr_data_t type.

```
#define r_imr_data_t T_IMR_DATA
typedef struct
{
   uint32_t Height;       /* Height of the picture(line) */
   uint32_t Width;        /* Width of the picture(pixel) */
   uint32_t bpp;          /* Color depth of the picture(bit/pixel) */
   uint32_t Stride;       /* Stride of the picture (pixel) */
   uint32_t Attr;         /* Attribute of the picutre */
   uint32_t PhysAddr;     /* Start address of the physical memory */
   Address  VmrStartAddr; /* Start address of the virtual memory area */
} T_IMR_DATA;
```

Figure 2.10 r_imr_data_t Type

2.2.6 r_imr_exec_opt_t

The r_imr_exec_opt_t type is a type that defines a format of the input image. Figure 2.11 shows the r_imr_exec_opt_t type.

```
#define r_imr_exec_opt_t T_IMRDRV_EXEC_OPT
typedef enum IMRDRV_EXEC_OPT
{
    R_IMR_EXEC_Y = 0,/*Perform rendering with the input image as Y image*/
    R_IMR_EXEC_UV,/*Perform rendering with the input image as UV image*/
} T_IMRDRV_EXEC_OPT;
```

Figure 2.11 r_imr_exec_opt_t Type

2.3 Macro Definitions

2.3.1 IMR_DATA_SIZE

The IMR_DATA_SIZE macro calculates the size (in bytes) of the image memory area from the r_imr_data_t type variable. Figure 2.12 shows the IMR_DATA_SIZE macro. Use this macro to calculate the size of the memory area specified in R_MMGR_AllocMemory ().

```
#define IMR_OCTET  (8U)
#define IMR_DATA_SIZE(d) ((((((d).Stride*(d).Height*(d).bpp) / IMR_OCTET) ¥
        + (R_MMGR_ALIGN_SIZE - 1U))/R_MMGR_ALIGN_SIZE)*R_MMGR_ALIGN_SIZE)
```

Figure 2.12 IMR_DATA_SIZE macro

2.3.2 IMR_DL_SIZE

The IMR_DL_SIZE macro calculates the size (in bytes) of the DL storage memory area from the number of DL elementes. Figure 2.13 shows the IMR_DL_SIZE macro. Use this macro to calculate the size of the memory area specified in R_MMGR_AllocMemory ().

```
#define IMR_DL_SIZE(d)  (((((d) * IMR_SIZEOF_DL_OPERAND) ¥
       + (R_MMGR_ALIGN_SIZE - 1U)) / R_MMGR_ALIGN_SIZE) * R_MMGR_ALIGN_SIZE)
```

Figure 2.13 IMR_DL_SIZE macro

2.3.3 Attribute macros

The attribute macros are definitions for specifying image attributes. These macros are set to Attr of the r_imr_data_t type. Figure 2.14 shows the attribute macros.

```
#define IMR_YUV        (0U)   /* Interleave format */
#define IMR_YUVSEP     (1U)   /* Separate format   */
```

Figure 2.14 Attribute macros

2.3.4 Return Values

Table 2.2 lists the macro definitions for return values of this software.

Table 2.2 Macro Definitions for Return Values

Definition Name	Value	Summary
E_OK	0	Successful completion
E_SYS	-5	System error
E_PAR	-17	Parameter error
E_OBJ	-41	State error
E_BOVR	-58	Buffer overflow

2.4 List of APIs

Table 2.3 lists the IMR driver APIs.

Table 2.3 List of APIs

API	Summary	Page
R_IMR_Init	Initializes IMR driver.	10
R_IMR_Deinit	Deinitialize IMR driver.	10
R_IMR_Start	Starts IMR driver.	11
R_IMR_Stop	Stop IMR driver.	11
R_IMR_Open	Opens IMR driver.	12
R_IMR_Close	Closes IMR driver.	12
R_IMR_CreateDl	Create DL area.	13
R_IMR_DestroyDl	Destroy DL area.	13
R_IMR_SetSource	Sets source image.	14
R_IMR_SetDstClippingArea	Specifies clip area.	15
R_IMR_SoftwareReset	Resets software.	16
R_IMR_CoordinateTransform	Transforms coordinates.	17
R_IMR_FillRect	Fills rectangle.	18
R_IMR_DlTri	Writes TRI instruction.	19
R_IMR_DlNop	Writes NOP instruction.	20
R_IMR_DlTrap	Writes TRAP instruction.	21
R_IMR_DlWtl	Writes WTL instruction.	21
R_IMR_DlWtl2	Writes WTL2 instruction.	22
R_IMR_DlWts	Writes WTS instruction.	23
R_IMR_DlInt	Writes INT instruction.	23
R_IMR_DlSyncm	Writes SYNCM instruction.	24
R_IMR_DlGosub	Writes GOSUB instruction.	25
R_IMR_DlRet	Writes RET instruction.	25
R_IMR_ClearDl	Clears DL storage.	26
R_IMR_RewindDl	Initializes DL write counter.	26
R_IMR_ExecuteExt	Executes DL.	27
R_IMR_WaitEvent	Waits for the completion of DL execution.	29
R_IMR_WaitEventTimeout	Waits for the completion of DL execution with timeout.	30

2.4.1 APIs for Driver Initialization

This subsection describes the driver initialization-related API specifications.

(1) R_IMR_Init

R_IMR_Init		
Summary	Initialize IMR driver.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_Init(void);	
Description	This API initializes the data area to be internally used by the IMR driver. It changes the driver software state of all the channels to the “Stop” state. E_OBJ is returned when the driver software state is other than “Uninitialization”. The driver software state will not be changed when the error is returned.	
Argument	None	
Return Value	E_OK	: Successful completion
	E_OBJ	: Driver software state error

(2) R_IMR_Deinit

R_IMR_Deinit		
Summary	Deinitialize IMR driver.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_Deinit(void);	
Description	This API changes the dirver software state of all the channels to the “Uninitialization” state. E_OBJ is returned when the driver software state is other than “Stop”. The driver software state will not be changed when the error is returned.	
Argument	None	
Return Value	E_OK	: Successful completion
	E_OBJ	: Driver software state error

(3) R_IMR_Start

R_IMR_Start		
Summary	Start IMR driver.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_Start(const int32_t channel);	
Description	This API changes the driver software state of an IMR module specified by the argument channel from the “Stop” state to the “Close” state that is able to be opened. E_PAR is returned when the argument channel is out of range, and E_OBJ is returned when the driver software state is other than “Stop”. The driver software state will not be changed when the error is returned.	
Argument	const int32_t channel	Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_OBJ	: Driver software state error

(4) R_IMR_Stop

R_IMR_Stop		
Summary	Stop IMR driver.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_Stop(const int32_t channel);	
Description	This API changes the driver software state of an IMR module specified by the argument channel from the “Close” state to the “Stop” state. E_PAR is returned when the argument channel is out of range, and E_OBJ is returned when the driver software state is other than “Close”. The driver software state will not be changed when the error is returned.	
Argument	const int32_t channel	Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_OBJ	: Driver software state error

(5) R_IMR_Open

R_IMR_Open	
Summary	Open IMR driver.
Header	r_imr_api.h
Declaration	int32_t R_IMR_Open(const int32_t channel);
Description	This API sets the channel of an IMR module specified by the argument channel to the state that is able to be used and changes the driver software state to the “Open” state. It also sets the channel of the specified IMR module to be able to be interrupted. E_PAR is returned when the argument channel is out of range, and E_OBJ is returned when the driver software state is other than “Close”. E_SYS is returned when the OS service call executed within this API returns error. The driver software state will not be changed when the error is returned.
Argument	const int32_t channel Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	E_OK : Successful completion E_PAR : Parameter error E_OBJ : Driver software state error E_SYS : System error

(6) R_IMR_Close

R_IMR_Close	
Summary	Close IMR driver.
Header	r_imr_api.h
Declaration	int32_t R_IMR_Close(const int32_t channel);
Description	This API performs the close processing on an IMR module specified by the argument channel and changes the driver software state to the “Close” state. It also sets the channel of the specified IMR module to be unable to be interrupted. E_PAR is returned when the argument channel is out of range, and E_OBJ is returned when the driver software state is other than the “Open” state. The driver software state will not be changed when E_PAR or E_OBJ is returned. E_SYS is returned when the OS service call executed within this API returns error. The driver software state will be changed to “Cose” in response to E_SYS.
Argument	const int32_t channel Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	E_OK : Successful completion E_PAR : Parameter error E_OBJ : Driver software state error E_SYS : System error

2.4.2 Reservation and Discard of DL Storage

This subsection describes the API specifications for reserving/discarding the DL storage.

(1) R_IMR_CreateDl

R_IMR_CreateDl	
Summary	Create DL storage.
Header	r_imr_api.h
Declaration	int32_t R_IMR_CreateDl(r_imr_dl_t * const dl);
Description	This API initializes the DL storage area specified by the argument dl for use by the IMR driver. For dl -> Size, specify the size (bytes) of the DL storage area. For dl -> PhysAddr, specify the physical address of the DL storage area. For dl -> VmrStartAddr, specify the virtual address of the DL storage area. Dl -> Pos is used by the IMR driver. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
Argument	r_imr_dl_t * const dl DL area
Return Value	E_OK : Successful completion E_PAR : Parameter error

(2) R_IMR_DestroyDl

R_IMR_DestroyDl	
Summary	Destroy DL storage.
Header	r_imr_api.h
Declaration	int32_t R_IMR_DestroyDl(r_imr_dl_t * const dl);
Description	This API clears the DL storage area specified by the argument dl E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
Argument	r_imr_dl_t * const dl DL area
Return Value	E_OK : Successful completion E_PAR : Parameter error

2.4.3 DL Operation Settings

This subsection describes the API specifications for setting the DL operation.

(1) R_IMR_SetSource

R_IMR_SetSource		
Summary	Set source image.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_SetSource(const int32_t channel, const r_imr_source_t * const src_setting);	
Description	This API sets the source image information for rendering on a channel of the IMR module specified by the argument channel. The argument src_setting uses the members source_width and source_height only. As the member start_line, end_line, or mesh_size will not be used, setting values can be arbitrary. E_PAR is returned if the argument channel is invalide, the argument src_setting is NULL, or the value set to source_width or source_height is out of range. Shown below are the setting ranges. 2 ≤ source_width ≤ 2048 2 ≤ source_height ≤ 2048 E_OBJ is returned if the driver software state is other than “Open”. This API does not change the driver software state.	
Argument	const int32_t channel	Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
	const r_imr_source_t * const src_setting	Source image settings
Return value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_OBJ	: Driver software state error

(2) R_IMR_SetDstClippingArea

R_IMR_SetDstClippingArea

Summary

Set clip area.

Header

r_imr_api.h

Declaration

```
int32_t R_IMR_SetDstClippingArea(const int32_t channel,
        r_imr_dl_t * const dl,
        const uint16_t x_min, const uint16_t y_min,
        const uint16_t x_max, const uint16_t y_max);
```

Description

This API writes the instruction to set the clip area into the clip register of IMR module channel specified by the argument channel in the DL storage specified by the argument dl. The clip area is specified by the rectangle area with the upper left coordinates (x_min, y_min) and the lower right coordinates (x_max, y_max).

The specified clip area is not directly set to the register, but the setting value is set to the register when executing the DL in order to write the register rewrite instruction into the DL.

When using fixed points for the destination coordinates, use the fixed points with 0 set to lower 2 bits of the decimal part for the coordinates of the clip area.

Specify the clip area before executing TRI instruction in order not to output the rendering result outside the destination area.

The number of DL elements to be written into the DL storage by this API is five.

E_BOVR is returned if the DL storage is insufficient.

E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.

E_PAR is returned if the channel number is invalid or x_min, y_min, x_max or y_max is out of range.

E_OBJ is returned if the driver software state is other than “Open”.

This API does not change the driver software state.

Argument

const int32_t channel	Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
r_imr_dl_t * const dl	DL area
const uint16_t x_min	Minimum value of X coordinate (0 - 0x1FFC)
const uint16_t y_min	Minimum value of Y coordinate (0 - 0x1FFC)
const uint16_t x_max	Maximum value of X coordinate (0 - 0x1FFC)
const uint16_t y_max	Maximum value of Y coordinate (0 - 0x1FFC)

Return Value

E_OK	: Successful completion
E_OBJ	: Driver software state error
E_PAR	: Parameter error
E_BOVR	: Insufficient DL storage

(3) R_IMR_SoftwareReset

R_IMR_SoftwareReset	
Summary	Reset software.
Header	r_imr_api.h
Declaration	int32_t R_IMR_SoftwareReset(const int32_t channel);
Description	This API performs the software reset on the channel of IMR module specified by the argument channel. E_PAR is returned if channel is out of range. E_OBJ is returnd if the driver software state is other than “Open”. This API does not change the driver software state.
Argument	const int32_t channel Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	E_OK : Successful completion E_PAR : Parameter error E_OBJ : Driver software state error E_SYS : System error

2.4.4 DL Operation

The DL instruction shall be written into the DL storage is created by the function R_IMR_CreateDL, and that DL storage shall be specified for the function R_IMR_ExecuteExt in order to perform the rendering by IMR.

The DL operation APIs are APIs to write the DL instruction into the DL area. By using them, DL elements corresponding to APIs will be written. The IMR driver software state will not be cahnged by the execution of DL operation APIs.

This subsection shows the API specifications for the DL operation.

(1) R_IMR_CoordinateTransform

R_IMR_CoordinateTransform		
Summary	Transform coordinates.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_CoordinateTransform(const r_imr_2dpos_t * const src, const float_t matrix[3][3], r_imr_2dpos_t * const dst, const int32_t num);	
Description	This API performs the affine transformation on the source coordinates specified by the argument src with the matrix specified by the argument matrix and stores the result in the area specified by the argument dst. The number of coordinates to be transformed will be specified by num. E_PAR is returned if the argument src is NULL, the argument matrix is NULL the argument dst is NULL or the argument num is 0 or less.	
Argument	const r_imr_2dpos_t * const src	Source coordinate array
	const float_t matrix[3][3]	Affin transformation matrix (3x3)
	r_imr_2dpos_t * const dst	Coordinate storage destination for the transformation result
	const int32_t num	Number of coordinates
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error

(2) R_IMR_FillRect

R_IMR_FillRect

Summary
Fill rectangle.

Header
r_imr_api.h

Declaration
int32_t R_IMR_FillRect(const uint32_t color, const r_imr_rect_t * const rect, r_imr_dl_t * const dl);

Description
This API writes the DL instruction for filling the rectangle area rect with the color specified by the argument color into the DL storage specified by the argument dl. Set the argument color to the setting values to the TRICR register. Shown below are the setting ranges of rect.

- 32768 ≤ rect->X ≤ 32766
- 32768 ≤ rect->Y ≤ 32766
1 ≤ rect->Width ≤ 65535
1 ≤ rect->Height ≤ 65535
rect->X + rect->Width ≤ 32767
rect->Y + rect->Height ≤ 32767

- YCFORM bit setting values of the argument color
Y/UV separation format: Fixed to 0.
Interleaved format : Output in YUYV order. Set 0.
Output in UYVY order. Set 1.

This API writes the DL instructions to set the TRIMR register, AMXSR register, AMYSR register, AMXOR register, AMYOR register.
Source / destination coordinates should be an integer.
E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
E_PAR is returned if the argument rect is NULL or the argument rect is out of range.
The number of DL elements to be written into the DL storage by this API is max 2154. E_BOVR is returned if the DL storage is insufficient.

Argument

const uint32_t color	Setting values to the TRICR register Bit 31: YCFORM bit Bits 30 to 24: Set 0. Bits 23 to 16: V setting value Bits 15 to 8: U setting value Bits 7 to 0: Y setting value For the bit numbers, MSB shall be bit 31, and LSB shall be bit 0.
const r_imr_rect_t * const rect	Specify a filled rectangle area.
r_imr_dl_t * const dl	DL area

Return Value

E_OK	: Successful completion
E_PAR	: Parameter error
E_BOVR	: Insufficient DL storage

(3) R_IMR_DlTri

R_IMR_DlTri

Summary

Write TRI instruction.

Header

r_imr_api.h

Declaration

```
int32_t R_IMR_DlTri(r_imr_dl_t * const dl, const uint16_t vertex_num,
                    const r_imr_2dpos_t * const src, const r_imr_2dpos_t * const dst,
                    const _Bool relative_pos)
```

Description

This API writes the TRI instruction with the argument src as the source coordinates and the argument dst as the destination coordinates and the DL instruction to set whether the automatic generation of coordinates is enabled or not into the DL storage specified by the argument dl. The number of coordinates is specified in the argument vertex_num.

Set NULL to the argument src when setting the source coordinates to be automatically generated. Set NULL to the argument dst when setting the destination coordinates to be automatically generated. When the automatic generation is set, the coordinates of mesh size specified in the IMR registers AMXSR and AMYSR will be generated starting from the coordinages set to the registers AMXOR and AMYOR. Figure 2.15 shows the example of automatic generation. The registers AMXOR, AMYOR, AMXSR, and AMYSR are not set by this API. Add the instruction for setting the target register to the DL when specifying the automatic generation coordinates by this API.

E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
E_PAR is returned if the argument vertex_num is out of range, both arguments src and dst are NULL or relative_pos is ture.

The number of DL elements to be written into the DL storage by this API is as follows:

When using the automatic geneation mode of coodinates: 4 + vertex_num
When specifying the coordinates: 4 + vertex_num * 2

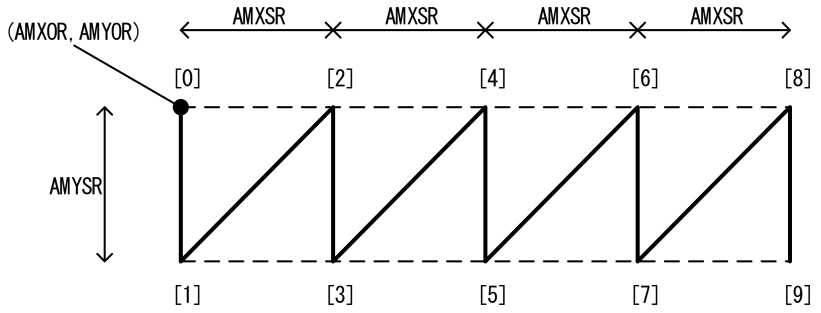
E_BOVR is returned if the DL storage is insufficient.

Argument

r_imr_dl_t * const dl	DL area
const uint16_t vertex_num	Number of coordinates (3 to 65535)
const r_imr_2dpos_t * const src	Source coordinate array
const r_imr_2dpos_t * const dst	Destination coordinate array
const _Bool relative_pos	Relative coordinate flag Specify false.

Return Value

E_OK	: Successful completion
E_PAR	: Parameter error
E_BOVR	: Insufficient DL storage



Starting from the coordinates set to the AMXOR and AMYOR registers, coordinates are generated in a zig-zag manner in the right direction with the width and height set to the AMXSR and AMYSR registers.
This figure is an example of automatic generation when the number of coordinates is 10 (vertex_num = 10).

Figure 2.15 Example of Automatic Generation of Coordinates

(4) R_IMR_DlNop

R_IMR_DlNop

Summary Write NOP instruction.

Header r_imr_api.h

Declaration int32_t R_IMR_DlNop(r_imr_dl_t * const dl, const uint16_t count);

Description This API adds the NOP (No Operation) instruction fo the DL storage specified by the argument dl. The NOP instruction executes the number of cycles specified by the argument count.

E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
E_PAR is returned if the argument count is 0.

The number of DL elements to be added to the DL storage by this API is one.
E_BOVR is returned if the DL storage is insufficient.

Argument

r_imr_dl_t * const dl	DL area
const uint16 count	Number of NOP instruction cycles (≥1)

Return Value

E_OK	: Successful completion
E_PAR	: Parameter error
E_BOVR	: Insufficient DL storage

(5) R_IMR_DlTrap

R_IMR_DlTrap		
Summary	Write TRAP instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlTrap(r_imr_dl_t * const dl);	
Description	This API adds the TRAP instruction to the DL storage specified by the argument dl. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. The number of DL elements to be added to the DL storage by this API is one. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(6) R_IMR_DlWtl

R_IMR_DlWtl		
Summary	Write WTL instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlWtl(r_imr_dl_t * const dl, const uint16_t start_reg_addr, const uint16_t num, const uint32_t * const data);	
Description	This API adds the WTL (Write Register Long) instruction to the DL storage specified by the argument dl. The WTL instruction writes data into as many registers as the number of nums starting with the register specified by the register address start_reg_addr. The lower 16 bits of the addresses of the write destination IMR registers are specified for the argument start_reg_addr. However, the valid values as address offsets are 0x0000 to 0x03FC. The array of data to be written is specified for the argument data. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. E_PAR is returned if the argument start_reg_addr is not 4-byte aligned, or it is 0x400 or more. E_PAR is also returned if the argument num is 0 or the argument data is NULL. The number of DL elements to be added to the DL storage by this API is 1+num. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
	const uint16_t start_reg_addr	Lower 16 bits of the address of write destination register
	const uint16_t num	Number of data to be written (1 to 65535)
	const uint32_t * const data	Array of data to be written
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(7) R_IMR_DlWtl2

R_IMR_DlWtl2

Summary

Write WTL2 instruction.

Header

r_imr_api.h

Declaration

```
int32_t R_IMR_DlWtl2(r_imr_dl_t * const dl, const uint16_t start_reg_addr,
                    const uint16_t num, const uint32_t * const data);
```

Description

This API adds the WTL2 (Write Register Long2) instruction to the DL storage sepcified by the argument dl. The WTL2 instruction writes data into as many registers as the number of nums starting with the register specified by the register address start_reg_addr.

The lower 16 bits of the addresses of the write destination IMR registers are specified for the argument start_reg_addr. Unlike the WTL instruction, there is no limitation of address offsets. The array of data to be written is specified for the argument data.

E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
E_PAR is returned if the argument start_reg_addr is not 4-byte aligned. E_PAR is also returned if the argument num is out of range or the argument data is NULL.

The number of DL elements to be added to the DL storage by this API is 1+num.
E_BOVR is returned if the DL storage is insufficient.

Argument

r_imr_dl_t * const dl	DL area
const uint16_t start_reg_addr	Lower 16 bits of the address of write destination register
const uint16_t num	Number of data to be written (1 to 512)
const uint32_t * const data	Array of data to be written

Return Value

E_OK	: Successful completion
E_PAR	: Parameter error
E_BOVR	: Insufficient DL storage

(8) R_IMR_DlWts

R_IMR_DlWts		
Summary	Write WTS instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlWts(r_imr_dl_t * const dl, const uint16_t reg_addr, const uint16_t data);	
Description	This API adds the WTS (Write Register Short) instruction to the DL storage specified by the argument dl. The lower 16 bits of the addresses of the write destination IMR registers are specified for the argument reg_addr. However, the valid values as address offsets are 0x0000 to 0x03FC. The data to be written is specified for the argument data. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. E_PAR is returned if the argument reg_addr is not 4-byte aligned, or it is 0x400 or more. The number of DL elements to be added to the DL storage by this API is one. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
	const uint16_t reg_addr	Lower 16 bits of the address of write destination register
	const uint16_t data	Data to be written
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(9) R_IMR_DlInt

R_IMR_DlInt		
Summary	Write INT instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlInt(r_imr_dl_t * const dl);	
Description	This API adds the INT (Interrupt) instruction to the DL storage specified by the argument dl. When the IMR decodes the INT instruction, the rendering is continued by clearing the status in this driver. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. The number of DL elements to be added to the DL storage by this API is one. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(10) R_IMR_DlSyncm

R_IMR_DlSyncm	
Summary	Write SYNCM instruction.
Header	r_imr_api.h
Declaration	int32_t R_IMR_DlSyncm(r_imr_dl_t * const dl, const _Bool fbs);
Description	This API adds the SYNCM instruction to the DL storage specified by the argument dl. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. The number of DL elements to be added to the DL storage by this API is one. E_BOVR is returned if the DL storage is insufficient.
Argument	r_imr_dl_t * const dl DL area const _Bool fbs Unused. Specify false.
Return Value	E_OK : Successful completion E_PAR : Parameter error E_BOVR : Insufficient DL storage

(11) R_IMR_DlGosub

R_IMR_DlGosub		
Summary	Write GOSUB instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlGosub(r_imr_dl_t * const dl, const Address addr);	
Description	This API adds the GOSUB (Go Subroutine) instruction to the DL storage specified by the argument dl. The address of the branch destination is specified for the argument addr. Specify the branch destination address by the virtual address in the DL storage specified by dl. Set the branch destination address addr and the starting address in the DL storage to multiples of 8. Otherwise, E_PAR is returned. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. E_PAR is returned if the argument addr is 0. The number of DL elements to be added to the DL storage by this API is two. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
	const Address addr	Branch destination address
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(12) R_IMR_DlRet

R_IMR_DlRet		
Summary	Write RET instruction.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_DlRet(r_imr_dl_t * const dl);	
Description	This API adds the RET (Return) instruction to the DL storage specified by the argument dl. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4. The number of DL elements to be added to the DL storage by this API is one. E_BOVR is returned if the DL storage is insufficient.	
Argument	r_imr_dl_t * const dl	DL area
Return Value	E_OK	: Successful completion
	E_PAR	: Parameter error
	E_BOVR	: Insufficient DL storage

(13) R_IMR_ClearDl

R_IMR_ClearDl	
Summary	Clear DL storage.
Header	r_imr_api.h
Declaration	int32_t R_IMR_ClearDl(r_imr_dl_t * const dl);
Description	This API clears the DL storage specified by the argument dl to 0 and initializes the write counter to 0. The written DL instruction becomes invalid. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
Argument	r_imr_dl_t * const dl DL area
Return Value	E_OK : Successful completion E_PAR : Parameter error

(14) R_IMR_RewindDl

R_IMR_RewindDl	
Summary	Initialize DL write counter.
Header	r_imr_api.h
Declaration	int32_t R_IMR_RewindDl(r_imr_dl_t * const dl);
Description	This API initializes the write counter to the DL storage specified by the argument dl to 0. The written DL instruction becomes invalid. E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0, dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
Argument	r_imr_dl_t * const dl DL area
Return Value	E_OK : Successful completion E_PAR : Parameter error

2.4.5 DL Execution

This subsection describes APIs for executing the DL and confirming the completion of processing.

(1) R_IMR_ExecuteExt

R_IMR_ExecuteExt

Summary

Execute DL.

Header

r_imr_api.h

Declaration

```
int32_t R_IMR_ExecuteExt(const int32_t channel, const r_imr_dl_t * const dl,
        const r_imr_data_t * const src, const r_imr_data_t * const dst,
      const uint32_t mode, const r_imr_exec_opt_t opt);
```

Description

This API executes the DL specified by the argument dl using the IMR module channel specified by the argument channel. Source image is specified for the argument src while memory region for storing transformation results is specified for the argument dst. Use a 256-byte aligned memory region for the argument src while use a 64-byte aligned memory region for the argument dst. E_PAR is returned if the alignment is invalid.

Specify flags shown in Table 2.4 for the argument mode by logical OR. Set the value of the argument mode to the TRIMR register before executing the DL. The TRIMR register can be overwritten by the WTS, WTL, or WTL2 instruction described in the DL.

If the source image is a Y/UV separation image, specify the following parameters according to the source image for the argument opt.
- R_IMR_EXEC_Y: Brightness
- R_IMR_EXEC_UV: Color difference

The argument opt will not be used if the source image is an interleaved image (if the memory attribute Attr is IMR_YUV).

E_OBJ is returned when the driver software state is other than “Open”.
The driver software state is changed to “DL running” in response to the successful completion. It is not changed in response to the error completion.
E_PAR is returned if the argument dl is NULL, dl -> PhysAddr = 0,
dl -> VmrStartAddr = 0, dl -> PhysAddr is not 8-byte aligned or dl -> Size <4.
E_PAR is returned if the argument src is NULL, src -> PhysAddr = 0 or
src -> VmrStartAddr = 0.
E_PAR is returned if the arrugment dst is NULL, dst -> PhysAddr = 0 or
dst -> VmrStartAddr = 0.
E_PAR is returned if the stride of the source image is less than 256 bytes, it is greater than 8192 bytes or it is not 256-byte aligned. E_PAR is returned if the stride of the destination image is less than 64 bytes, it is greater than 8192 bytes or it is not 64-byte aligned.

Argument

const int32_t channel	Channel numbers (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
const r_imr_dl_t * const dl	DL area
const r_imr_data_t * const src	Source image data
const r_imr_data_t * const dst	Destination imagedata
const uint32_t mode	TRIMR register settings
const r_imr_exec_opt_t opt	Y/UV select

Return Value

E_OK	: Successful completion
E_OBJ	: Driver software state error
E_PAR	: Parameter error

Table 2.4 Flags for Setting TRIMR Register

Flag	Specified	Not Specified
R_IMR_EXE_MODE_CLOCKWISE	Triangle clockwise mode	Triangle counterclockwise mode
R_IMR_EXE_MODE_AUTOSRC	Source coordinates automatic generation mode is enabled.	Source coordinate automatic generation mode is disabled.
R_IMR_EXE_MODE_AUTODST	Destination coordinates automatic generation is enabled.	Destination coordinates automatic generation mode is disabled.
R_IMR_EXE_MODE_BILINEAR_ENABLE	Bilinear filtering is used.	Biliear filtering is not used.
R_IMR_EXE_MODE_TEXTUREMAPPING	Texture mapping is used.	Texture mapping is not used.

(2) R_IMR_WaitEvent

R_IMR_WaitEvent	
Summary	Wait for the completion of DL execution.
Header	r_imr_api.h
Declaration	int32_t R_IMR_WaitEvent(int32_t channel);
Description	This API waits for the completion of DL execution on the IMR module channel specified by the argument channel. If the driver software is in the “DL running” state, this API will be in the wait state within the function until an interrupt that completes the DL execution occurs. When the interrupt occurs, it releases the wait state and returns the status information at the occurrence of interrupt shown in Table 2.5. It also chages the driver software state to “Open”. The wait state is not released if the INT instruction is executed. Interrupt occurence events: (1) TRAP instruction is executed. (2) Invalid DL instruction is executed. If this API is executed when the driver software is in the “Open” state, it will not be in the wait state within the function and return the status information on the occurrence of the previous interrupt. R_IMR_SR_ERR is returned if the argument channel is out of range, and the driver software state is “Uninitialization”, “Stop” or “Close”, an error is returned without changing the driver software state. R_IMR_SR_ERR is returned without changing the driver software state when the OS service call returns error. The status information on the return value is 0 if the function R_IMR_ExecuteExt is not executed.
Argunemt	int32_t channel Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
Return Value	Status bit (Table 2.5)

Table 2.5 Status bit

Status bit	Set	Clear
R_IMR_SR_REN (0x0000 0020)	DL is in execution.	DL is not in execution.
R_IMR_SR_IER (0x0000 0002)	Invalid DL instruction is decoded.	Invalid DL instruction is not decoded.
R_IMR_SR_TRA (0x0000 0001)	TRAP instruction is decoded.	TRAP instruction is not decoded.
R_IMR_SR_ERR (0x8000 0000)	Error is occurred.	Error is not occurred.

(3) R_IMR_WaitEventTimeout

R_IMR_WaitEventTimeout		
Summary	Wait for the completion of DL execution with timeout.	
Header	r_imr_api.h	
Declaration	int32_t R_IMR_WaitEventTimeout(const int32_t channel, const uint32_t tmout);	
Description	This API waits for the completion of DL execution on the IMR module channel specified by the argument channel. If the driver software is in the “DL running” state, this API will be in the wait state within the function until an interrupt that completes the DL execution occurs. When the interrupt occurs, it releases the wait state and returns the status information at the occurrrence of interrupt. It also changes the driver software state to “Open”. The wait state is not released if the INT instruction is executed. Interrupt occurrence events: (1) TRAP instruction is executed. (2) Invalid DL instruction is executed. The wait state of this API is released, and R_IMR_SR_ERR is returned if the interrupt does not occur after the time specified by the argument tmout passes. The driver software state is changed to “Open”. This timeout assumes that IMR - LX 4 can not render normally. For the wait time specified by the argument tmout, set a sufficient time for the DL execution to complete. When timeout occurs, execute the R_IMR_Close function and R_IMR_Stop function and return the driver software state to ”Stop”. If this API is executed when the driver software is in the “Open” state, it will not be in the wait state within the function and return the status information on the occurrence of the previous interrupt. R_IMR_SR_ERR is returned if the argument channel is out of range, and the driver software state is “Uninitialization” ,“Stop ” or “Close ”, an error is returned without changing the driver software state. R_IMR_SR_ERR is returned without changing the driver software state when the OS service call returns error. The status information on the return value is 0 if the function R_IMR_ExecuteExt is not executed.	
Argument	const int32_t channel	Channel number (R-Car H3: 0 to 3, R-Car M3-W: 0, 1)
	const uint32_t tmout	Wait time (msec)
Return Value	Status bit	(Table 2.5)

2.5 Notes

2.5.1 Execution by Multitasking

This software does not control exclusive control. When executing API from multiple tasks, please perform exclusive control with user application.

2.5.2 DL Operation

This software does not perform the cache operation. Therefore, after writing the DL instruction to the DL storage, coherency between the cache and physical memory should be guaranteed by the user application.

Also, do not rewrite or release the DL storage that is performing the rendering. If rewrote or released, the behavior would be undefined.

Shown below are the API functions that manipulate the DL storage.

- R_IMR_CreateDL function
- R_IMR_DestroyDl function
- R_IMR_SetDstClippingArea function
- R_IMR_FillRect function
- R_IMR_DlTri function
- R_IMR_DlNop function
- R_IMR_DlTrap function
- R_IMR_DlWtl function
- R_IMR_DlWtl2 function
- R_IMR_DlWts function
- R_IMR_DlInt function
- R_IMR_DlSyncm function
- R_IMR_DlGosub function
- R_IMR_DlRet function
- R_IMR_ClearDl function
- R_IMR_RewindDl function

Revision History	R-Car Series, 3rd Generation IMR Device Driver for INTEGRITY ® User’s Manual: Software

Rev.	Date	Description	
		Page	Summary
0.50	Jan 11, 2017	—	First Edition issued
0.51	Apr 13,2017	3	Update Figure 2.3 State Transition Diagram.
		8	Add 2.3.3 Attribute macros
		15	Update Argument of 2.4.3 (2) R_IMR_SetDstClippingArea.
		16	Update Description of 2.4.3 (3) R_IMR_SoftwareReset
		18	Update Description of 2.4.4 (2) R_IMR_FillRect
		27	Update Description of 2.4.5 (1) R_IMR_ExecuteExt
		28	Update Table 2.4 Flags for Setting TRIMR Register
		30	Update Description of 2.4.5 (3) R_IMR_WaitEventTimeout
		ALL	Modification of the sentence

R-Car Series, 3rd Generation
IMR Device Driver for INTEGRITY ®
User’s Manual: Software

Publication Date: Rev.0.51 Apr 13, 2017

Published by: Renesas Electronics Corporation

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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

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

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

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

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

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

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

Colophon 4.1

ルネサスエレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

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

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

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

Colophon 3.1

R-Car Series, 3rd Generation
IMR Device Driver for INTEGRITY ®

Renesas Electronics Corporation