RE01 1500KB、256KB Group

CMSIS Driver GDT Specification

Summary

This document describes the detailed specifications of the GDT driver provided in the RE01 CMSIS software package (hereinafter called the GDT driver). This driver uses GDT to do graphic processing.

Target Device

RE01 1500KB Group

RE01 256KB Group

Contents

1. Overview

1.1 Overview of the GDT driver

GDT driver uses 2 DMAC channels and performs the following graphic processing operations.

- Shrink (this operation is called “Reduction” in User’s Manual: Hardware)
- Endian conversion
- Monochrome synthesis
- Rotation
- Scroll
- Font development
- No change (Flip)
- Color data sorting
- Color synthesis
- Colorization

1.2 How to use this driver

This section explains the overview of how to use this driver. Detailed explanation is described in Section 6.1.

In this section, image processing steps are explained.

Step 1: Prepare the original image information.

1. Size of the image: Horizontal_size x Vertical_size

   This example uses 176 X 176 pixels. Therefore, the data is defined as uint8_t img_data[(176/8)*176].

2. Starting address of the image: Image start address

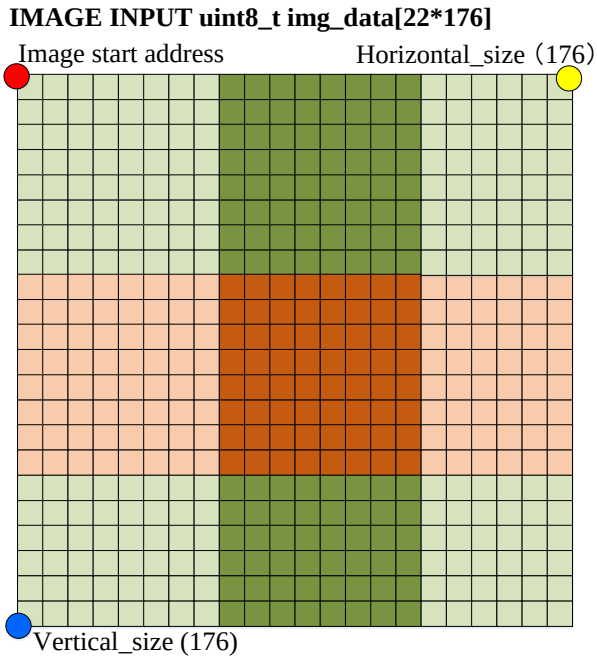

Figure 1-1 original image information

Step 2: Copy the original image from step 1 to the input buffer named IMAGE_BUF_SRC.

This driver has size restriction for the horizontal size of the buffer. GDT driver can only process images with the horizontal size of 32, 64, 128, or 256 pixels. If the image from Step 1 has different horizontal size, the data must be copied to the permitted size before GDT driver can run. See Section 5.2 for more information.

Figure 1-2 is the buffer image of Step 1 image with the size of 256 X 176 pixels (32*176 bytes)

1. Input buffer horizontal size: Memory_size
2. Size of the image: Horizontal_size x Vertical_size
3. Starting address of the input buffer: &IMAGE_BUF_SRC

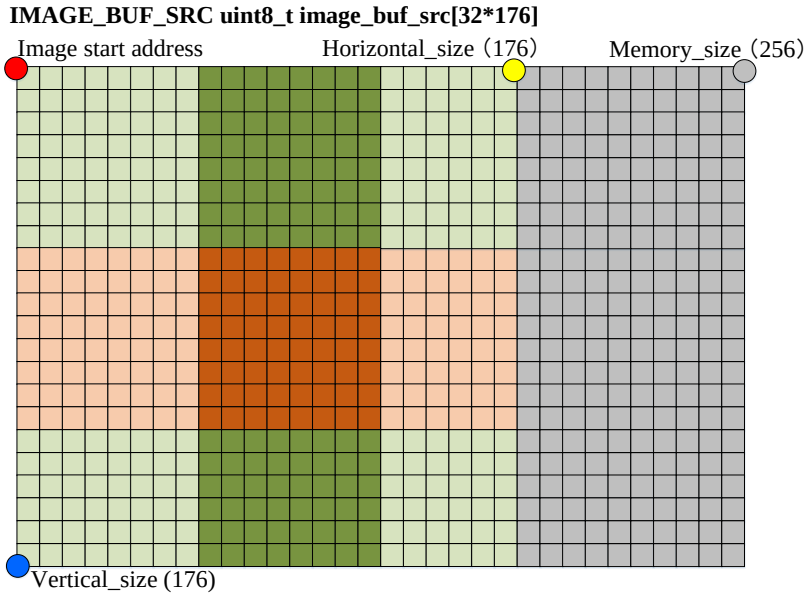

Figure 1-2 input buffer image information

Srep 3: Convert image data using GDT driver.

This step sets block information before GDT processing. Input buffer starting address from Step 2 and destination buffer starting address are inputted to GDT driver.

Block processing area is also selected:

1. Starting pixel coordinates of the processing block: (H,V)
2. Horizontal size of the processing block: horizontal_size
3. Vertical size of processing block: vertical_size

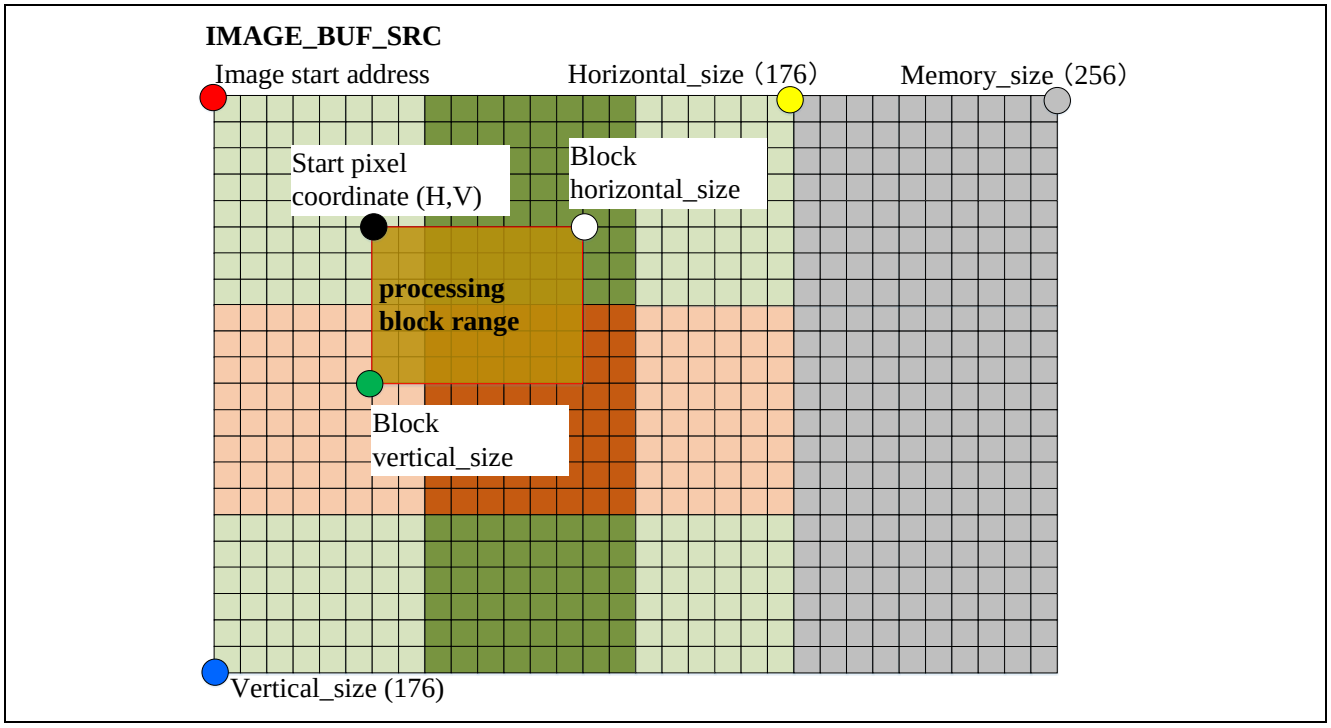

Figure 1-3 block information

The followings are destination image and block information after GDT processing.

1. Destination buffer horizontal size: Memory_size
2. Size of the image: Horizontal_size x Vertical_size
3. Starting address of the destination buffer: &IMAGE_BUF_DEST
4. Starting coordinates of the processed block: (H,V)

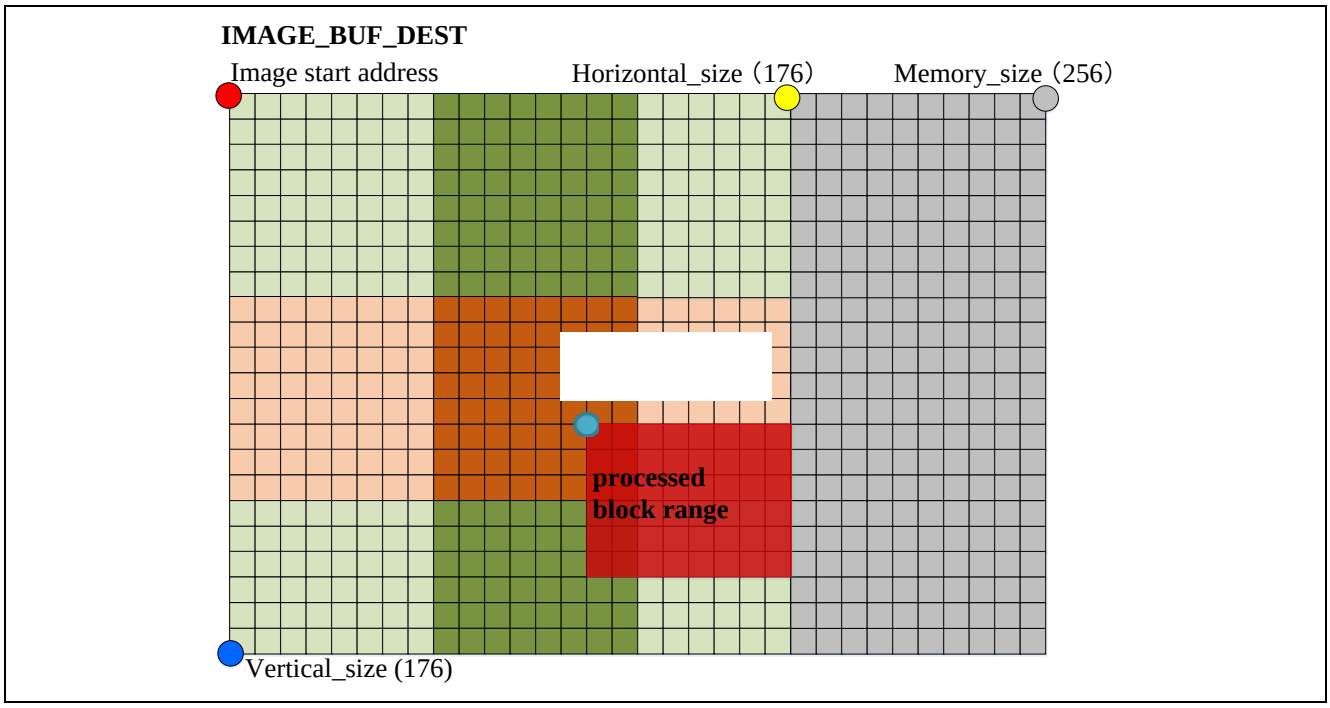

Figure 1-4 destinations image and block information

1.3 API List

Table 1-1 lists the API functions included in this driver.

Table 1-1 API Functions List

Function	Description
R_GDT_GetVersion	Get the driver version
R_GDT_Open	Enable GDT driver
R_GDT_Close	Start clock supply to GDT H/W
R_GDT_DmacChSel	Disable GDT driver
R_GDT_Shrink	Stop clock supply to GDT H/W
R_GDT_Endian	Select 2 DMAC channels to input data to GDT and output data from GDT
R_GDT_Monochrome	Shrink
R_GDT_Rotate	Endian conversion
R_GDT_Scroll	Monochrome Image synthesis
R_GDT_Fount	Rotation
R_GDT_Nochange	Scroll
R_GDT_Coloralign	Font development (glyph data conversion)
R_GDT_Colorsyn	
R_GDT_Color	Image data not changed

1.4 Processing Example

Figure 1-5 shows an example of calling functions from multiple APIs.

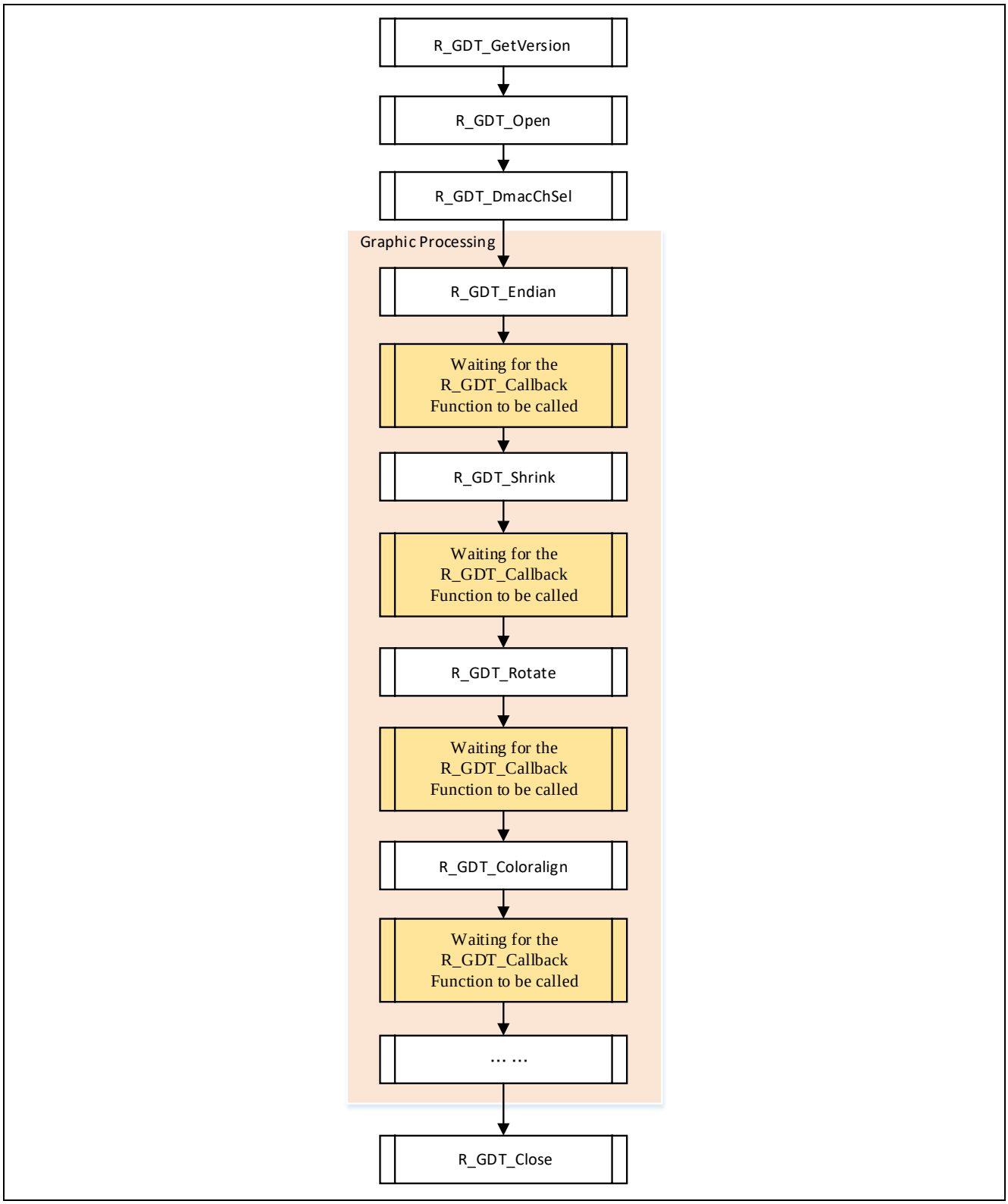

Figure 1-5 Example of calling multiple APIs

2. API Information

This driver has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- DMAC (Use 2 DMAC channels to input data to GDT and output data from GDT)
- GDT

2.2 Software Requirements

This driver was developed based on the following CMSIS driver:

- RE01 1500KB, 256KB Group CMSIS software package

2.3 Supported Toolchain

This driver has been confirmed to work with the following toolchain.

- GNU Arm Embedded Toolchain: v6-2017-q2-update
- 1500KB group : IAR Embedded Workbench Version 8.32.1
- 256KB group : IAR Embedded Workbench Version 8.40.2

2.4 Interrupt Vector

The GDT_DATII interrupt and GDT_DATOI interrupt are enabled by execution of v_gdt_imgdata_trans_irq_enable() function.

Two DMAC channels are required. The DMACx_INT interrupt are enabled by execution of R_DMACn_InterruptEnable() function.

Please refer to Table 2-1 for the interrupt vectors used in GDT driver.

Table 2-1 Interrupt Vector used in the GDT Driver

Device	Interrupt Vector
RE01 1500KB RE01 256KB	**Three interrupts of GDT driver are used.** GDT_DATII interrupt (event no.: A8h) GDT_DATOI interrupt (event no.: A9h) **Choose two DMAC channels from 4 available channels.** DMAC0_INT interrupt (event no.: 09h) DMAC1_INT interrupt (event no.: 0Ah) DMAC2_INT interrupt (event no.: 0Bh) DMAC3_INT interrupt (event no.: 0Ch) Notice: Set the DMAC interrupt priority level to satisfy the following conditions. (Change the interrupt definition in r_dmac_cfg.h) DMAC interrupt priority for data input to GDT > DMAC channel interrupt priority for data output

Example: Performing Shrink function. DMAC0 is used for sending GDT input data and DMAC1 is used for sending GDT output data.

①Setting SYSTEM_IRQ_EVENT_NUMBER (r_system_cfg.h)

```
#define  SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT  (SYSTEM_IRQ_EVENT_NUMBER0)
#define  SYSTEM_CFG_EVENT_NUMBER_DMAC1_INT  (SYSTEM_IRQ_EVENT_NUMBER5)
#define  SYSTEM_CFG_EVENT_NUMBER_GDT_DATOI  (SYSTEM_IRQ_EVENT_NUMBER4)
#define  SYSTEM_CFG_EVENT_NUMBER_GDT_DATII  (SYSTEM_IRQ_EVENT_NUMBER3)
```

②Setting the DMAC channel.

```
st_trans_mod_cfg_t -> ch_in          = DMAC0;
st_trans_mod_cfg_t -> ch_out         = DMAC1;
e_gdt_err_t  R_GDT_DmacChSel (st_trans_mod_cfg_t);
```

ICU event link setting parameters are shown in Table 2-2. DMAC event link setting parameters are shown in Table 2-3.

Table 2-2 ICU event link setting parameters

NO.	Interrupt name	Register(IELSR)	IELS[4:0]
1	GDT_DATII	IELSR 3/11/19/27	5'b11110
2	GDT_DATOI	IELSR 4/12/20/28	5'b11110
3	DMAC0	IELSR 0/4/8/12/16/20/24/28	5'b00010
4	DMAC1	IELSR 1/5/9/13/17/21/25/29	5'b00010
5	DMAC2	IELSR 2/6/10/14/18/22/26/30	5'b00010
6	DMAC3	IELSR 3/7/11/15/19/23/27/31	5'b00010

Table 2-3 DMAC event link setting parameters

NO.	Interrupt event link	Register（DELSR）	DELS[7:0]	channel
1	GDT_DATII	DELSR0	168	DMAC0
		DELSR1		DMAC1
		DELSR2		DMAC2
		DELSR3		DMAC3
2	GDT_DATOI	DELSR0	169	DMAC0
		DELSR1		DMAC1
		DELSR2		DMAC2
		DELSR3		DMAC3

2.5 Header Files

All API calls and their supporting interface definitions are located in r_gdt_api.h.

2.6 Configuration

The configuration option settings of this module are located in r_gdt_cfg.h. The option names and setting values are listed in the table below:

Table 2-4 Configuration information

Configuration options in r_gdt_cfg.h		Setting range	Initial value
GDT_CFG_COLOR_ON	Enable color data processing	0: monochrome 1: color (default) To reduce the memory usage, this option should be set to 0 when using monochrome processing.	1
GDT_CFG_IN_INT_PRIORITY	DATII interrupt priority setting	0: Set the priority to 0 (highest). 1: Set the priority to 1. 2: Set the priority to 2. 3: Set the priority to 3	2
GDT_CFG_OUT_INT_PRIORITY	DATOI interrupt priority setting	0: Set the priority to 0 (highest) 1: Set the priority to 1. 2: Set the priority to 2. 3: Set the priority to 3	3

2.7 Code Size

Typical code sizes associated with this module for RE01 1500KB Group is listed in Table 2-5.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in Section 2.6, Configuration. The table lists reference values when the C compiler's compile options are set to their default values, as described in Section 2.3, Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options. The code size of the GDT driver is shown in Table 2-5.

Table 2-5 GDT Driver Code Size

ROM, RAM, and Stack Code Sizes				
Device	Category	Memory Used		Remarks
		All functions allocated to RAM	All functions allocated to ROM	
RE01 1500KB	ROM	4856 bytes	19856 bytes	
	RAM	140528 bytes	125416 bytes	
	Maximum stack usage	620 bytes		Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

*GDT driver code size was confirmed under the following conditions:

Using six image buffers, with the size of 256x256bit each. Data is transmitted with 2 DMAC channels.

2.8 Parameters

This section describes the parameter structure used by the API functions in this module. The structure below is the prototype declarations of API functions and is located in r_gdt_api.h.

```
img_src *                   /* source image information */
        ->img_num,                              /* the number of source images*/
        ->start_addr[IMG_IN_FRAME_NUM],         /* source image address */
        ->size_h [IMG_IN_FRAME_NUM],            /* source image horizontal size */
        ->size_v [IMG_IN_FRAME_NUM],            /* source image vertical size */
        ->mem_size_h[IMG_IN_FRAME_NUM]          /* source image memory horizontal size */
img_dest*                   /* destination image information */
        ->img_num,                              /* the number of destination image */
        ->start_addr[IMG_OUT_FRAME_NUM],        /* destination image address */
        ->size_h[IMG_OUT_FRAME_NUM],            /* destination image horizontal size */
        ->size_v[IMG_OUT_FRAME_NUM],            /* destination image vertical size */
        ->mem_size_h[IMG_OUT_FRAME_NUM]         /* destination image memory horizontal size */
blk_conv_info*          /* information of processing block */
        ->start_pix_h_src[IMG_IN_FRAME_NUM],    /* start pixel of source image, horizontal direction */
        ->start_pix_v_src[IMG_IN_FRAME_NUM],    /* start pixel of source image, vertical direction */
        ->start_pix_h_dest[IMG_OUT_FRAME_NUM],
                                        /* start pixel of destination image, horizontal direction */
        ->start_pix_v_dest[IMG_OUT_FRAME_NUM], /* start pixel of destination image, vertical direction */
        ->size_h,                               /* the size of modification area, horizontal direction */
        ->size_v                                /* the size of modification area, vertical direction */
gdt_shrink_para_cfg*    /* GDT shrink function setting */
        ->iflp_en,                              /* flip function enable */
        ->shrnk_size,                           /* shrink size select */
        ->shrnk_bak_grnd_colr                   /* set the color for remain area */

trans_mod_cfg           /* select transfer mode */
        ->ch_in                                 /* input transfer channel select */
        ->ch_out                                /* output transfer channel select */

p_callback              /* callback function*/
```

2.9 Return Values

This section describes return values of API functions. This enumeration is the prototype declarations of API functions and is located in r_gdt_api.h.

```
typedef     enum(
        GDT_OK,                         /* successful operation */
        GDT_ERROR                       /* unspecified error */
        GDT_ERROR_IMG_CFG,              /* image configuration error */
        GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
        GDT_ERROR_BLK_CFG,              /* source or destination block size configuration error */
        GDT_ERROR_MOD_CFG,              /* graphic processing configuration error */
```

```
    GDT_ERROR_IMG_NUM,          /* Image number configuration error */
    GDT_ERROR_DMAC_CH_CFG       /* DMAC channel configuration error */
) gdt_err_t
```

Table 2-6 Return value

return type	description	solution
GDT_OK	The GDT is functioning normally	-
GDT_ERROR	1. GDT clock fails to start	Please refer to R_LPM_ModuleStart()
	2. GDT clock fails to stop	Please refer to R_LPM_ModuleStop()
	3. GDT_COLOR_ON definition has not been set when using color API.	Please modify r_gdt_cfg.h #define GDT_CFG_COLOR_ON (1)
GDT_ERROR_IMG_CFG	Incorrect memory size of input and output images.	Please confirm the images memory size setting information. Memory size limit (32,64,128,256) Coloralign function: Output image memory size limit 3bit mode(96,192,384,768) 4bit mode(128,256,512,1024)
	Incorrect horizontal size of input and output images.	Horizontal size is not a multiple of 8 Horizontal size is less than 8 Horizontal size is larger than memory size Vertical size is not a multiple of 8 Vertical size is less than 8
GDT_ERROR_BLK_OFF_BOUND	The input and output block starting point coordinates are out of range.	Please confirm the block starting coordinate setting information.
	The input and output block ending point coordinates are out of range.	Please confirm the block starting coordinate and block size setting information.
GDT_ERROR_BLK_CFG	The block starting point coordinates are not a multiple of 8	Please confirm the clocks setting information. Blocks horizontal size is not a multiple of 8 Blocks horizontal size is less than 8
	Incorrect horizontal size of the input and output blocks.	
GDT_ERROR_MOD_CFG	Incorrect GDTDSZ bit setting	Please confirm that the gdt_monochrome_para_cfg->gdtdsz is set correctly.
	Incorrect ISCREN bit setting of the scroll function.	The ISCREN bit is not allowed to be set to 0.
	R_GDT_Fount: the fdltdsz setting is not 7~63	Please confirm that the GDT register is set correctly.
	R_GDT_Fount: the fdlngsz setting is not 7~64	
GDT_ERROR_IMG_NUM	R_GDT_Monochrome: border function enable: input image number 3, output image number 1 border function disable: input image number 2, output image number 1	Please confirm that the number of input and output image is set correctly.
	R_GDT_Coloralign: input image number 3, output image number 1	
	R_GDT_Color: input image number 1, output image number 1	
	R_GDT_Colorsyn: input image number 6, output image number 3	
	normal function: input image number 1, output image number 1	
GDT_ERROR_DMAC_CH_CFG	Selected DMAC channel is not available	Select an available DMAC channel.
	DMAC channel setting error, setting range 0~3	Please confirm that the DMAC channel is set correctly.

2.10 Callback Function

In this module, the callback function specified by the user is called when the GDT processing is completed.

If there is user processing required after GDT conversion is completed, describe the process in the callback function.

```
/* Callback function usage example */
void R_GDT_Callback(void)
{
/* Customer code */
GDT_end_flag = 1; /* example GDT processing completed */
…
}
```

3. API Functions specification

3.1 R_GDT_Open()

The function initializes GDT and start clock supply to GDT. After calling this API, other GDT APIs can be used.

Format

e_gdt_err_t R_GDT_Open ( )

Parameters

None.

Return Values

```
typedef enum(
        GDT_OK,                         /* successful operation */
        GDT_ERROR                       /* unspecified error */
        GDT_ERROR_IMG_CFG,              /* image configuration error */
        GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
        GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
        GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
        GDT_ERROR_IMG_NUM,              /* Image number configuration error */
        GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”

Description

Performs the initialization to start the GDT H/W.

Set MSTPCRC26 bit and enable GDT clock.

Reentrant

No.

Example

None.

Special Notes:
None.

3.2 R_GDT_Close ()

The function terminates GDT driver. Clock supply to GDT H/W is stopped and power consumption is reduced.

Format
e_gdt_err_t R_GDT_Close ( )

Parameters
None.

Return Values

```
typedef enum(
        GDT_OK,                     /* successful operation */
        GDT_ERROR                   /* unspecified error */
        GDT_ERROR_IMG_CFG,          /* image configuration error */
        GDT_ERROR_BLK_OFF_BOUND,    /* block out of bounds error */
        GDT_ERROR_BLK_CFG,          /* source or destination block size configuration error */
        GDT_ERROR_MOD_CFG,          /* Graphic processing configuration error */
        GDT_ERROR_IMG_NUM,          /* Image number configuration error */
        GDT_ERROR_DMAC_CH_CFG       /* DMAC channel configuration error */
) gdt_err_t
```

Properties
Prototyped in “r_gdt_api.h”

Description
Set MSTPCRC26 bit and disable GDT clock.

Reentrant
No.

Example
None.

Special Notes:
None.

3.3 R_GDT_GetVersion()

The function gets the version of GDT API.

Format

```
uint32_t        R_GDT_GetVersion ( )
```

Parameters
None.

Return Values
Version number

Properties
Prototyped in file “r_gdt_api.h”

Description
This function will return the version of the currently installed GDT driver. The 2 high bytes are the major version number and the 2 low bytes are the minor version number. For example, Version 0.72 would be returned as 0x00000048.

Reentrant
No.

Example
None.

Special Notes:
None.

3.4 R_GDT_DmacChSel()

Select the DMAC channel according to the parameters.

Format

```
uint32_t        R_GDT_DmacChSel (
                st_trans_mod_cfg_t*   trans_mod_cfg/* select transfer mode */
)
```

Parameters

```
trans_mod_cfg           /* select transfer mode */
```

Return Values

```
typedef    enum(
                GDT_OK,                         /* successful operation */
                GDT_ERROR                       /* unspecified error */
                GDT_ERROR_IMG_CFG,              /* image configuration error */
                GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
                GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
                GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
                GDT_ERROR_IMG_NUM,              /* Image number configuration error */
                GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties
Prototyped in file “r_gdt_api.h”

Description
Select the DMAC channel according to the parameters, the channel will transfer data from and to GDT H/W.

- Checking the parameters
- Selecting the DMAC channel
- Checking the DMAC channel availability

Reentrant
No.

Example
None.

Special Notes:
For restrictions for the interrupt level setting of the DMAC used for GDT, see the Section 5.4 DMAC Interrupt Settings and Priority Limitations.

3.5 R_GDT_Shrink ()

The function shrinks image.

Format

```
e_gdt_err_t   R_GDT_Shrink (
              st_img_in_info_t*          img_src,              /* source image information */
              st_img_out_info_t*         img_dest,             /* destination image information */
              st_blk_conv_info_t*        blk_conv_info,        /* information of processing block */
              st_gdt_shrink_para_cfg_t*  gdt_shrink_para_cfg , /* GDT shrink function setting */
              gdt_cb_event_t const       p_callback            /* callback function*/
)
```

Parameters

```
img_src *                  /* source image information */
img_dest*                  /* destination image information */
blk_conv_info*             /* information of processing block */
gdt_shrink_para_cfg*       /* GDT shrink function setting */
        ->iflp_en                  /* flip function enable */
        ->shrnk_size               /* shrink size select */
        ->shrnk_bak_grnd_colr      /* set the color for remain area */
p_callback                 /* callback function*/
```

Return Values
typedef enum(

```
    GDT_OK,                         /* successful operation */
    GDT_ERROR                       /* unspecified error */
    GDT_ERROR_IMG_CFG,              /* image configuration error */
    GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
    GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
    GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
    GDT_ERROR_IMG_NUM,              /* Image number configuration error */
    GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the shrink mode configuration, the selected area of image will be shrunk and outputted to the target position. The shrink ratio is selectable from 1/8~7/8.

Shrink setting:

iflp_en: 0/1

shrnk_size: 0~7

shrnk_bak_grnd_colr: 0/1

This API processing flow are described by the following steps.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the shrink mode
- Starting the transmission sequence
- Enabling the GDT interrupts
- Graphic processing completed
- Call the callback function

Reentrant

No.

Example processing overview

Mode configuration: 7/8 shrink ratio, flip function ON, 0 redundant background color

For the shrink processing, one input image and one output image are used.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinates of (24,40) and size of (horizontal_size 152, vertical_size 32) from input image will be processed using shrink function. The result is then stored to output image with the starting coordinate of (24,40).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_shrink_para_cfg*    gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = &image_buf_src;
img_src_example->size_h[0]                  = 191;
img_src_example->size_v[0]                  = 127;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = &image_buf_dest;
img_dest_example->size_h[0]                 = 191;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 24;
blk_conv_info_example->start_pix_v_src[0]   = 40;
blk_conv_info_example->start_pix_h_dest[0]  = 24;
blk_conv_info_example->start_pix_v_dest[0]  = 40;
blk_conv_info_example->size_h               = 152;
blk_conv_info_example->size_v               = 32;
gdt_mod_para_cfg_example->iflp_en           = 1;
gdt_mod_para_cfg_example->shrnk_size        = 6; /* 7/8 shrink ratio */
gdt_mod_para_cfg_example->shrnk_bak_grnd_colr = 0;

e_gdt_err_t = R_GDT_shrink (
      img_src_example,
     img_dest_example,
     blk_conv_info_example,
     gdt_mod_para_cfg_example,
     p_callback*
)
```

Special Notes:

None.

3.6 R_GDT_Endian()

The function converts endianness of the image.

Format

```
e_gdt_err_t   R_GDT_Endian (
              st_img_in_info_t*          img_src,          /* source image information */
              st_img_out_info_t*         img_dest,         /* destination image information */
              st_blk_conv_info_t*        blk_conv_info,    /* information of processing block */
              st_gdt_endian_para_cfg_t*  gdt_endian_para_cfg,
                                                           /* endian conversion function setting */
```

```
        gdt_cb_event_t const    p_callback      /* callback function*/
)
```

Parameters

```
img_src *               /* source image information */
img_dest*               /* destination image information */
blk_conv_info*          /* information of processing block */
gdt_endian_para_cfg*    /* endian conversion function setting */
    ->iflp_en               /* flip function enable */
    ->endian                /* endian conversion function enable */
p_callback              /* callback function*/
```

Return Values

```
typedef  enum(
        GDT_OK,                         /* successful operation */
        GDT_ERROR                       /* unspecified error */
        GDT_ERROR_IMG_CFG,              /* image configuration error */
        GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
        GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
        GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
        GDT_ERROR_IMG_NUM,              /* Image number configuration error */
        GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the endian conversion mode configuration, the endianness of the selected area of image will be converted and outputted to the target position.

The data processing unit of the endian function is 16x16bits.

Endian conversion setting:
iflp_en: 0/1

endian: 0/1

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the endian mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: Endian ON, flip function ON

One input image and one output image are used for endian processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinates of (24,40) and size of (horizontal_size 80, vertical_size 32) from input image will be processed using endian conversion function. The result is then stored to output image with the starting coordinate of (24, 40).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_endian_para_cfg*    gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = 0x20004000;
img_src_example->size_h[0]                  = 191;
img_src_example->size_v[0]                  = 127;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = 0x20008000;
img_dest_example->size_h[0]                 = 191;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 24;
blk_conv_info_example->start_pix_v_src[0]   = 40;
blk_conv_info_example->start_pix_h_dest[0]  = 40;
blk_conv_info_example->start_pix_v_dest[0]  = 40;
blk_conv_info_example->size_h               = 80;
blk_conv_info_example->size_v               = 32;
gdt_mod_para_cfg_example->iflp_en           = 1;
gdt_mod_para_cfg_example->endian            = 1;
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_endian (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_endian_para_cfg_example,
      p_callback*
)
```

Special Notes:

None.

3.7 R_GDT_Nochange()

The function is used to flip the image.

Format

```
e_gdt_err_t   R_GDT_Nochange (
              st_img_in_info_t*              img_src,              /* source image information */
              st_img_out_info_t*             img_dest,             /* destination image information */
              st_blk_conv_info_t*            blk_conv_info,        /* information of processing block */
              st_gdt_nochange_para_cfg_t*    gdt_nochange_para_cfg, /*GDT nochange function setting*/
              gdt_cb_event_t const           p_callback            /* callback function*/
)
```

Parameters

```
img_src *                   /* source image information */
img_dest*                   /* destination image information */
blk_conv_info*              /* information of processing block */
gdt_nochange_para_cfg*      /* GDT nochange function setting */
      ->iflp_en                   /* flip function enable */
p_callback                  /* callback function*/
```

Return Values

```
typedef   enum(
            GDT_OK,                         /* successful operation */
            GDT_ERROR                       /* unspecified error */
            GDT_ERROR_IMG_CFG,              /* image configuration error */
            GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
            GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
            GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
            GDT_ERROR_IMG_NUM,              /* Image number configuration error */
            GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

The R_GDT_Nochange API calls the R_GDT_Endian() API.

Based on the input picture information and the nochange mode configuration, the selected area of image will be flipped and outputted to the target position.

The data processing unit of the endian conversion function is 16x16bits.

Nochange setting:

iflp_en: 0/1

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the nochange mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example

About image size, please refer the R_GDT_Endian() function for the detail.

Example processing overview

Mode configuration: Endian conversion OFF, flip function ON

One input image and one output image are used in nochange processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinates of (24,40) and size of (horizontal_size 80, vertical_size 32) from input image will be processed using nochange function. The result is then stored to output image with the starting coordinate of (24, 40).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_nochange_para_cfg*  gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = 0x20004000;
img_src_example->size_h[0]                  = 191;
img_src_example->size_v[0]                  = 127;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = 0x20008000;
img_dest_example->size_h[0]                 = 191;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 24;
blk_conv_info_example->start_pix_v_src[0]   = 40;
blk_conv_info_example->start_pix_h_dest[0]  = 40;
blk_conv_info_example->start_pix_v_dest[0]  = 40;
blk_conv_info_example->size_h               = 80;
```

```
blk_conv_info_example->size_v                = 32;
gdt_mod_para_cfg_example->iflp_en            = 1;
trans_mod_cfg_example->ch_in                 = 0;
trans_mod_cfg_example->ch_out                = 1;

e_gdt_err_t R_GDT_Nochange (
       img_src_example,
       img_dest_example,
       blk_conv_info_example,
       gdt_nochange_para_cfg_example,
       p_callback*
)
```

Special Notes:

None.

3.8 R_GDT_Monochrome()

The function performs monochrome synthesis.

Format

```
e_gdt_err_t    R_GDT_Monochrome (
               st_img_in_info_t*              img_src,       /* source image information */
               st_img_out_info_t*             img_dest,      /* destination image information */
               st_blk_conv_info_t*            blk_conv_info, /* information of processing block */
               st_gdt_monochrome_para_cfg_t*  gdt_monochrome_para_cfg,
                                                             /* GDT monochrome function setting */
               gdt_cb_event_t const    p_callback            /* callback function*/
)
```

Parameters

```
img_src *                    /* source image information */
img_dest*                    /* destination image information */
blk_conv_info*               /* information of processing block */
gdt_monochrome_para_cfg*     /* monochrome synthesis function setting */
      ->iflp_en                    /* flip function enable */
     ->gdtdsz                      /* processing data size selection 0:16*16bit; 1:8*8bit */
     ->mpcs                        /* monochrome priority color specification */
     ->mbrden                      /* border function enable */
p_callback                   /* callback function*/
```

Return Values

```
typedef    enum(
               GDT_OK,                    /* successful operation */
               GDT_ERROR                  /* unspecified error */
               GDT_ERROR_IMG_CFG,         /* image configuration error */
```

```
    GDT_ERROR_BLK_OFF_BOUND,      /* block out of bounds error */
    GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
    GDT_ERROR_MOD_CFG,          /* Graphic processing configuration error */
    GDT_ERROR_IMG_NUM,          /* Image number configuration error */
    GDT_ERROR_DMAC_CH_CFG       /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the monochrome synthesis mode configuration, monochrome synthesis processing will be performed on the selected area of image and outputted to the target position.

The data processing unit of the monochrome synthesis function is 8x8bits.

Monochrome synthesis setting:
iflp_en: 0/1

gdtdsz: 0/1

mpcs: 0/1

mbrden: 0/1

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the monochrome mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: flip function ON, 1 synthesis, border function enabled, graphics processing data size 8x8bit.

Three input images and one output image are used for monochrome synthesis processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC[i], size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinates of (24,40) (48, 40) (120, 32) and size of (horizontal_size 64, vertical_size 32) from input image will be processed using monochrome synthesis function. The result is then stored to output image with the starting coordinate of (64, 40).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
```

```
img_dest*                img_dest_example;
blk_conv_info*           blk_conv_info_example;
gdt_monochrome_para_cfg* gdt_mod_para_cfg_example;
trans_mod_cfg*           trans_mod_cfg_example;

img_src_example->img_num                    = 3;
img_src_example->start_addr[0]              = &image_buf_src[0];
img_src_example->start_addr[1]              = &image_buf_src[1];
img_src_example->start_addr[2]              = &image_buf_src[2];
img_src_example->size_h                     = 191;
img_src_example->size_v                     = 127;
img_src_example->mem_size_h                 = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = &image_buf_dest;
img_dest_example->size_h                    = 191;
img_dest_example->size_v                    = 127;
img_dest_example->mem_size_h                = 256;
blk_conv_info_example->start_pix_h_src[0]   = 24;
blk_conv_info_example->start_pix_v_src[0]   = 24;
blk_conv_info_example->start_pix_h_src[1]   = 48;
blk_conv_info_example->start_pix_v_src[1]   = 40;
blk_conv_info_example->start_pix_h_src[2]   = 120;
blk_conv_info_example->start_pix_v_src[2]   = 32;
blk_conv_info_example->start_pix_h_dest[0]  = 64;
blk_conv_info_example->start_pix_v_dest[0]  = 64;
blk_conv_info_example->size_h               = 64;
blk_conv_info_example->size_v               = 32;
gdt_mod_para_cfg_example->iflp_en           = 1;
gdt_mod_para_cfg_example->gdtdsz            = 1;
gdt_mod_para_cfg_example->mpcs              = 1;
gdt_mod_para_cfg_example->mbrden            = 1;
trans_mod_cfg_example->mod_sel              = DMAC;
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_Monochrome (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_monochrome_para_cfg_example,
      trans_mod_cfg_example
      p_callback*
)
```

Special Notes:

In this example, the processed image data size is 8x8 bits. Images with 16x16 bits data size is not compatible with this API function.

3.9 R_GDT_Rotate()

The function rotates image.

Format

```
e_gdt_err_t   R_GDT_Rotate (
              st_img_in_info_t*     img_src,      /* source image information */
              st_img_out_info_t*    img_dest,     /* destination image information */
```

```
                st_blk_conv_info_t*         blk_conv_info,      /* information of processing block */
                st_gdt_rotate_para_cfg_t*   gdt_rotate_para_cfg, /* GDT rotate function setting */
                gdt_cb_event_t const        p_callback          /* callback function*/
  )
```

Parameters

```
  img_src *                 /* source image information */
  img_dest*                 /* destination image information */
  blk_conv_info*            /* information of processing block */
  gdt_rotate_para_cfg*      /* rotate function setting */
        ->iflp_en                 /* flip function enable */
        ->gdtdsz                  /* graphic processing data size selection 0:16*16bit; 1:8*8bit */
          ->rttfc                 /* rotate function select */
  p_callback                /* callback function*/
```

Return Values

```
  typedef     enum(
                GDT_OK,                         /* successful operation */
                GDT_ERROR                       /* unspecified error */
                GDT_ERROR_IMG_CFG,              /* image configuration error */
                GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
                GDT_ERROR_BLK_CFG,        /* source or destination block size configuration error */
                GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
                GDT_ERROR_IMG_NUM,              /* Image number configuration error */
                GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
  ) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the rotate mode configuration, the selected area of image will be rotated and outputted to the target position.

The data processing unit of the rotate function can be either 8x8bits or 16x16bits.

Rotation setting:
iflp_en: 0/1

gdtdsz: 0/1

rttfc: 0~3

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the rotate mode

- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: rotates 90° in clockwise direction, flip function ON, graphics processing data size 16x16bit.

One input image and one output image are used for rotation processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinate of (48,24) and size of (horizontal_size 32, vertical_size 64) from input image will be processed using rotate function. The result is then stored to output image with the starting coordinate of (48, 24).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_rotate_para_cfg*    gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = 0x20004000;
img_src_example->size_h[0]                  = 191;
img_src_example->size_v[0]                  = 127;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = 0x20008000;
img_dest_example->size_h[0]                 = 191;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 48;
blk_conv_info_example->start_pix_v_src[0]   = 24;
blk_conv_info_example->start_pix_h_dest[0]  = 48;
blk_conv_info_example->start_pix_v_dest[0]  = 24;
blk_conv_info_example->size_h               = 32;
blk_conv_info_example->size_v               = 64;
gdt_mod_para_cfg_example->iflp_en           = 1;
gdt_mod_para_cfg_example->gdtdsz            = 0;
gdt_mod_para_cfg_example->rttfc             = 0; /* 90° right*/
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_Rotate (
     img_src_example,
     img_dest_example,
     blk_conv_info_example,
     gdt_rotate_para_cfg_example,
     p_callback*
```

```
)
```

Special Notes:
None.

3.10 R_GDT_Scroll()

The function is used to scroll image.

Format

```
e_gdt_err_t    R_GDT_Scroll (
               st_img_in_info_t*          img_src,          /* source image information */
               st_img_out_info_t*         img_dest,         /* destination image information */
               st_blk_conv_info_t*        blk_conv_info,    /* information of processing block */
               st_gdt_scroll_para_cfg_t*  gdt_scroll_para_cfg,  /* scroll function setting */
               gdt_cb_event_t const       p_callback        /* callback function*/
)
```

Parameters

```
img_src *                  /* source image information */
img_dest*                  /* destination image information */
blk_conv_info*             /* information of processing block */
gdt_scroll_para_cfg*       /* GDT scroll function setting */
    ->iflp_en                  /* flip function enable */
    ->iscren                   /* scroll function enable */
p_callback                 /* callback function*/
```

Return Values

```
typedef    enum(
               GDT_OK,                          /* successful operation */
               GDT_ERROR                        /* unspecified error */
               GDT_ERROR_IMG_CFG,               /* image configuration error */
               GDT_ERROR_BLK_OFF_BOUND,         /* block out of bounds error */
               GDT_ERROR_BLK_CFG,       /* source or destination block size configuration error */
               GDT_ERROR_MOD_CFG,               /* Graphic processing configuration error */
               GDT_ERROR_IMG_NUM,               /* Image number configuration error */
               GDT_ERROR_DMAC_CH_CFG            /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the scroll mode configuration, the selected area of image will be scrolled and outputted to the target position.

The data processing unit of the scroll function is 16x16bits.

Scroll setting:
iflp_en: 0/1

iscren: 1~7 (0: scroll function disabled)

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the scroll mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: scrolling 1bit, flip function ON

One input image and one output image are used for scroll processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinate of (40,32) and size of (horizontal_size 128, vertical_size 16) from input image will be processed using scroll function. The result is then stored to output image with the starting coordinate of (40, 32).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_scroll_para_cfg*    gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              =  0x20004000;
img_src_example->size_h[0]                  = 191;
img_src_example->size_v[0]                  = 127;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = 0x20008000;
img_dest_example->size_h[0]                 = 191;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 40;
blk_conv_info_example->start_pix_v_src[0]   = 32;
blk_conv_info_example->start_pix_h_dest[0]  = 40;
blk_conv_info_example->start_pix_v_dest[0]  = 32;
blk_conv_info_example->size_h               = 128;
```

```
blk_conv_info_example->size_v                = 16;
gdt_mod_para_cfg_example->iflp_en            = 1;
gdt_mod_para_cfg_example->iscren             = 001; /* scroll 1 bit*/
trans_mod_cfg_example->ch_in                 = 0;
trans_mod_cfg_example->ch_out                = 1;

e_gdt_err_t R_GDT_Scroll (
        img_src_example,
        img_dest_example,
        blk_conv_info_example,
        gdt_scroll_para_cfg_example,
        p_callback*
)
```

Special Notes:

None.

3.11 R_GDT_Fount()

The function is used to expand the font (converting the glyph data stored in the RAM to a format that can be easily displayed on the LCD).

Format

```
e_gdt_err_t    R_GDT_Fount (
               st_img_in_info_t*          img_src,          /* source image information */
               st_img_out_info_t*         img_dest,         /* destination image information */
               st_blk_conv_info_t*        blk_conv_info,    /* information of processing block */
               st_gdt_fount_para_cfg_t*   gdt_fount_para_cfg, /* GDT fount function setting */
               gdt_cb_event_t const       p_callback        /* callback function*/
)
```

Parameters

```
img_src *                  /* source image information */
img_dest*                  /* destination image information */
blk_conv_info*             /* information of processing block */
gdt_fount_para_cfg*        /* fount function setting */
        ->iflp_en                  /* flip function enable */
        ->sac                      /* start address change bit */
        ->fdhad                    /* blank bit setting */
        ->fdltdsz                  /* font horizontal number of bit setting */
        ->iscren                   /* font vertical number of bit setting */
p_callback                 /* callback function*/
```

Return Values

```
typedef     enum(
                GDT_OK,                    /* successful operation */
                GDT_ERROR                  /* unspecified error */
```

```
    GDT_ERROR_IMG_CFG,          /* image configuration error */
    GDT_ERROR_BLK_OFF_BOUND,    /* block out of bounds error */
    GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
    GDT_ERROR_MOD_CFG,          /* Graphic processing configuration error */
    GDT_ERROR_IMG_NUM,          /* Image number configuration error */
    GDT_ERROR_DMAC_CH_CFG       /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the font mode configuration, the glyph data stored in the RAM will be converted into a format that can be easily displayed on the LCD.

Font setting:
iflp_en: 0/1

sac: 0~7

fdhad: 0/1

fdltdsz: 7~63

fdlngsz: 7/64

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the font mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: flip function ON, start address change 4 bit

The value of the unused bit is 0.

The font size is H10xV10

One input image and one output image are used for glyph data conversion processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H192xV128，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H192xV128，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinate of (0,0) and size of (horizontal_size 0, vertical_size 0) from input image will be processed using font expansion function. The result is then stored to output image with the starting coordinate of (8, 2).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_fount_para_cfg*     gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = 0x20004000;
img_src_example->size_h[0]                  = 0;
img_src_example->size_v[0]                  = 0;
img_src_example->mem_size_h[0]              = 0;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = 0x20008000;
img_dest_example->size_h[0]                 = 192;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 0;
blk_conv_info_example->start_pix_v_src[0]   = 0;
blk_conv_info_example->start_pix_h_dest[0]  = 8;
blk_conv_info_example->start_pix_v_dest[0]  = 2;
blk_conv_info_example->size_h               = 0;
blk_conv_info_example->size_v               = 0;
gdt_mod_para_cfg_example->iflp_en           = 0;
gdt_mod_para_cfg_example->sac               = 4;
gdt_mod_para_cfg_example->fdhad             = 0;
gdt_mod_para_cfg_example->fdltdsz           = 10;
gdt_mod_para_cfg_example->fdlngsz           = 10;
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_Fount (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_fount_para_cfg_example,
      p_callback*
)
```

Special Notes:

None.

3.12 R_GDT_Coloralign()

The function is used to perform color data sorting.

Format

```
e_gdt_err_t    R_GDT_Coloralign (
               st_img_in_info_t*              img_src,                /* source image information */
               st_img_out_info_t*             img_dest,               /* destination image information */
               st_blk_conv_info_t*            blk_conv_info,          /* information of processing block */
               st_gdt_coloralign_para_cfg_t*  gdt_coloralign_para_cfg, /* GDT coloralign function setting*/
               gdt_cb_event_t const    p_callback                     /* callback function*/
)
```

Parameters

```
img_src *                   /* source image information */
img_dest*                   /* destination image information */
blk_conv_info*              /* information of processing block */
gdt_coloralign_para_cfg*    /* GDT coloralign function setting */
        ->iflp_en               /* flip function enable */
        ->cialgsl               /* color data sorting mode sel */
p_callback                  /* callback function*/
```

Return Values

```
typedef     enum(
                GDT_OK,                         /* successful operation */
                GDT_ERROR                       /* unspecified error */
                GDT_ERROR_IMG_CFG,              /* image configuration error */
                GDT_ERROR_BLK_OFF_BOUND,        /* block out of bounds error */
                GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
                GDT_ERROR_MOD_CFG,              /* Graphic processing configuration error */
                GDT_ERROR_IMG_NUM,              /* Image number configuration error */
                GDT_ERROR_DMAC_CH_CFG           /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the color data sorting mode configuration, R data, G data and B data will be sorted and outputted to the target position.

The data processing unit of the color data sorting function is 16x16bits.

Color data sorting setting:
iflp_en: 0/1

cialgsl: 0/1

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the coloralign mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: flip function ON, 3-bit color data sorting mode.

Three input images and one output image are used for color data sorting processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H176xV176，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST, size of the image: H176xV176，memory horizontal size: 769.

Processing block setting:

The processing blocks (R, G, and B) with starting coordinate of (H16, V16) (H24, V40) (H80, V8) and size of (horizontal_size 32, vertical_size 32) from input image will be processed using color data sorting function. The result is then stored to output image with the starting coordinate of (48, 56).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_coloralign_para_cfg*gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 3;
img_src_example->start_addr[0]              = &image_buf_src[0];
img_src_example->start_addr[1]              = &image_buf_src[1];
img_src_example->start_addr[2]              = &image_buf_src[2];
img_src_example->size_h[0]                  = 176;
img_src_example->size_v[0]                  = 176;
img_src_example->size_h[1]                  = 176;
img_src_example->size_v[1]                  = 176;
img_src_example->size_h[2]                  = 176;
img_src_example->size_v[2]                  = 176;
img_src_example->mem_size_h[0]              = 256;
img_src_example->mem_size_h[1]              = 256;
img_src_example->mem_size_h[2]              = 256;
img_dest_example->img_num                   = 1;
img_dest_example->start_addr[0]             = &image_buf_dest;
img_dest_example->size_h[0]                 = 192;
img_dest_example->size_v[0]                 = 127;
img_dest_example->mem_size_h[0]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 16;
blk_conv_info_example->start_pix_v_src[0]   = 16;
blk_conv_info_example->start_pix_h_src[1]   = 24;
blk_conv_info_example->start_pix_v_src[1]   = 40;
blk_conv_info_example->start_pix_h_src[2]   = 80;
blk_conv_info_example->start_pix_v_src[2]   = 8;
blk_conv_info_example->start_pix_h_dest[0]  = 48;
blk_conv_info_example->start_pix_v_dest[0]  = 56;
blk_conv_info_example->size_h               = 32;
blk_conv_info_example->size_v               = 32;
gdt_mod_para_cfg_example->iflp_en           = 0;
gdt_mod_para_cfg_example-> cialgsl          = 0; /*RGB 3bit*/
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_Coloralign (
            img_src_example,
            img_dest_example,
            blk_conv_info_example,
```

```
            gdt_coloralign_para_cfg_example,
            p_callback*
)
```

Special Notes:
None.

3.13 R_GDT_Colorsyn()

The function is used for color data synthesis processing.

Format

```
e_gdt_err_t   R_GDT_Colosyn (
              st_img_in_info_t*          img_src,              /* source image information */
              st_img_out_info_t*         img_dest,             /* destination image information */
              st_blk_conv_info_t*        blk_conv_info,        /* information of processing block */
              st_gdt_colorsyn_para_cfg_t* gdt_colorsyn_para_cfg, /* GDT colorsyn function setting */
              gdt_cb_event_t const       p_callback            /* callback function*/
)
```

Parameters

```
img_src *                   /* source image information */
img_dest*                   /* destination image information */
blk_conv_info*              /* information of processing block */
gdt_colorsyn_para_cfg*      /* color synthesis function setting */
    ->iflp_en                   /* flip function enable */
    ->gdtdsz                    /* graphic processing data size selection 0:16*16bit; 1:8*8bit */
    ->cpts                      /* priority/transparency mode setting */
    ->cdcs                      /* specified color setting */
p_callback                  /* callback function*/
```

Return Values

```
typedef   enum(
              GDT_OK,                       /* successful operation */
              GDT_ERROR                     /* unspecified error */
              GDT_ERROR_IMG_CFG,            /* image configuration error */
              GDT_ERROR_BLK_OFF_BOUND,      /* block out of bounds error */
              GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
              GDT_ERROR_MOD_CFG,            /* Graphic processing configuration error */
              GDT_ERROR_IMG_NUM,            /* Image number configuration error */
              GDT_ERROR_DMAC_CH_CFG         /* DMAC channel configuration error */
) gdt_err_t
```

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the color synthesis mode configuration, the selected area of image will be synthesized and outputted to the target position.

color synthesis setting:
iflp_en: 0/1

gdtdsz:0/1

cpts: 0/1

cdcs:0~7

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the colorsyn mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: flip function ON, Graphics processing data size 16x16bit, priority color mode.

foreground color : red.

Six input images and three output images are used in this color synthesis processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC[i], size of the image: H176xV176， memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST[j], size of the image: H176xV176， memory horizontal size: 256.

Processing block setting:

In input images, the block starting coordinates of the foreground R block, the foreground G block, the foreground B block, the background R block, the background G block, the background B block are (H16, V16) (H16, V16) (H16, V16) (H32, V40) (H32, V40) (H32, V40). The processing block with the size of (horizontal_size 128, vertical_size 64) will be processed using color synthesis function. The result is then stored to output image with the starting coordinate of (48, 64).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_colorsyn_para_cfg*  gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 6;
img_src_example->start_addr[0]              = &image_buf_src[0];
img_src_example->start_addr[1]              = &image_buf_src[1];
img_src_example->start_addr[2]              = &image_buf_src[2];
```

```
img_src_example->start_addr[3]              = &image_buf_src[3];
img_src_example->start_addr[4]              = &image_buf_src[4];
img_src_example->start_addr[5]              = &image_buf_src[5];
img_src_example->size_h[0]                  = 176;
img_src_example->size_v[0]                  = 176;
img_src_example->size_h[1]                  = 176;
img_src_example->size_v[1]                  = 176;
img_src_example->size_h[2]                  = 176;
img_src_example->size_v[2]                  = 176;
img_src_example->size_h[3]                  = 176;
img_src_example->size_v[3]                  = 176;
img_src_example->size_h[4]                  = 176;
img_src_example->size_v[4]                  = 176;
img_src_example->size_h[5]                  = 176;
img_src_example->size_v[5]                  = 176;
img_src_example->mem_size_h[0]              = 256;
img_src_example->mem_size_h[1]              = 256;
img_src_example->mem_size_h[2]              = 256;
img_src_example->mem_size_h[3]              = 256;
img_src_example->mem_size_h[4]              = 256;
img_src_example->mem_size_h[5]              = 256;
img_dest_example->img_num                   = 3;
img_dest_example->start_addr[0]             = &image_buf_dest[0];
img_dest_example->start_addr[1]             = &image_buf_dest[1];
img_dest_example->start_addr[2]             = &image_buf_dest[2];
img_dest_example->size_h[0]                 = 176;
img_dest_example->size_v[0]                 = 176;
img_dest_example->size_h[1]                 = 176;
img_dest_example->size_v[1]                 = 176;
img_dest_example->size_h[2]                 = 176;
img_dest_example->size_v[2]                 = 176;
img_dest_example->mem_size_h[0]             = 256;
img_dest_example->mem_size_h[1]             = 256;
img_dest_example->mem_size_h[2]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 16;
blk_conv_info_example->start_pix_v_src[0]   = 16;
blk_conv_info_example->start_pix_h_src[1]   = 16;
blk_conv_info_example->start_pix_v_src[1]   = 16;
blk_conv_info_example->start_pix_h_src[2]   = 16;
blk_conv_info_example->start_pix_v_src[2]   = 16;
blk_conv_info_example->start_pix_h_src[3]   = 32;
blk_conv_info_example->start_pix_v_src[3]   = 40;
blk_conv_info_example->start_pix_h_src[4]   = 32;
blk_conv_info_example->start_pix_v_src[4]   = 40;
blk_conv_info_example->start_pix_h_src[5]   = 32;
blk_conv_info_example->start_pix_v_src[5]   = 40;
blk_conv_info_example->start_pix_h_dest[0]  = 48;
blk_conv_info_example->start_pix_v_dest[0]  = 64;
blk_conv_info_example->start_pix_h_dest[1]  = 48;
blk_conv_info_example->start_pix_v_dest[1]  = 64;
blk_conv_info_example->start_pix_h_dest[2]  = 48;
blk_conv_info_example->start_pix_v_dest[2]  = 64;
blk_conv_info_example->size_h               = 128;
blk_conv_info_example->size_v               = 64;
gdt_mod_para_cfg_example->iflp_en           = 0;
gdt_mod_para_cfg_example->gdtdsz            = 0;
gdt_mod_para_cfg_example->cpts              = 0;
gdt_mod_para_cfg_example->cdcs              = 4;
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;
```

```
e_gdt_err_t R_GDT_Colorsyn (
       img_src_example,
       img_dest_example,
       blk_conv_info_example,
       gdt_colorsyn_para_cfg_example,
       p_callback*
)
```

Special Notes:
None.

3.14 R_GDT_Color()

The function is used to turn monochrome image into color image.

Format

```
e_gdt_err_t    R_GDT_Color (
               st_img_in_info_t*          img_src,          /* source image information */
               st_img_out_info_t*         img_dest,         /* destination image information */
               st_blk_conv_info_t*        blk_conv_info,    /* information of processing block */
               st_gdt_color_para_cfg_t*   gdt_color_para_cfg, /* GDT color function setting */
               gdt_cb_event_t const       p_callback        /* callback function*/
)
```

Parameters

```
img_src *                  /* source image information */
img_dest*                  /* destination image information */
blk_conv_info*             /* information of processing block */
gdt_color_para_cfg*        /* GDT colorsyn function setting */
      ->iflp_en                  /* flip function enable */
      ->clrds0                   /* part 0 RGB color setting */
      ->clrds1                   /* part 1 RGB color setting */
p_callback                 /* callback function*/
```

Return Values

```
typedef    enum(
               GDT_OK,                      /* successful operation */
               GDT_ERROR                    /* unspecified error */
               GDT_ERROR_IMG_CFG,           /* image configuration error */
               GDT_ERROR_BLK_OFF_BOUND,     /* block out of bounds error */
               GDT_ERROR_BLK_CFG,      /* source or destination block size configuration error */
               GDT_ERROR_MOD_CFG,           /* Graphic processing configuration error */
               GDT_ERROR_IMG_NUM,           /* Image number configuration error */
               GDT_ERROR_DMAC_CH_CFG        /* DMAC channel configuration error */
```

) gdt_err_t

Properties

Prototyped in file “r_gdt_api.h”.

Description

Based on the input picture information and the colorization mode configuration, the selected area of image will be colorized and outputted to the target position.

colorization setting:
iflp_en: 0/1

clrds0: 0~7

clrds1:0~7

The following steps are the flow of this API processing.

- Checking the parameters
- Initializing variables used by the API
- Setting the priority of GDT interrupt
- Calculating the transmission information.
- Setting the register for the colorization mode
- Starting the transmission sequence
- Enabling the GDT interrupts

Reentrant

No.

Example processing overview

Mode configuration: flip function ON,

The color of “0” bit is changed to red (4).

The color of “1” bit is changed to blue (1).

One input image and three output images are used in colorization processing.

Input/output image and memory setting:

Input image is in the buffer named IMAGE_BUF_SRC, size of the image: H176xV176，memory horizontal size: 256.

Output image is in the buffer named IMAGE_BUF_DEST[j], size of the image: H176xV176，memory horizontal size: 256.

Processing block setting:

The processing block with starting coordinate of (24,40) and size of (horizontal_size 80, vertical_size 32) from input image will be processed using colorization function. The result is then stored to output images(R, G and B) with the starting coordinate of (40, 40).

Example code

```
return_value*           e_gdt_err_t;
img_src*                img_src_example;
img_dest*               img_dest_example;
blk_conv_info*          blk_conv_info_example;
gdt_colorsyn_para_cfg*  gdt_mod_para_cfg_example;
trans_mod_cfg*          trans_mod_cfg_example;

img_src_example->img_num                    = 1;
img_src_example->start_addr[0]              = &image_buf_src;
img_src_example->size_h[0]                  = 176;
img_src_example->size_v[0]                  = 176;
img_src_example->mem_size_h[0]              = 256;
img_dest_example->img_num                   = 3;
img_dest_example->start_addr[0]             = &image_buf_dest[0];
img_dest_example->start_addr[1]             = &image_buf_dest[1];
img_dest_example->start_addr[2]             = &image_buf_dest[2];
img_dest_example->size_h[0]                 = 176;
img_dest_example->size_v[0]                 = 176;
img_dest_example->size_h[1]                 = 176;
img_dest_example->size_v[1]                 = 176;
img_dest_example->size_h[2]                 = 176;
img_dest_example->size_v[2]                 = 176;
img_dest_example->mem_size_h[0]             = 256;
img_dest_example->mem_size_h[1]             = 256;
img_dest_example->mem_size_h[2]             = 256;
blk_conv_info_example->start_pix_h_src[0]   = 40;
blk_conv_info_example->start_pix_v_src[0]   = 40;
blk_conv_info_example->start_pix_h_dest[0]  = 40;
blk_conv_info_example->start_pix_v_dest[0]  = 40;
blk_conv_info_example->start_pix_h_dest[1]  = 40;
blk_conv_info_example->start_pix_v_dest[1]  = 40;
blk_conv_info_example->start_pix_h_dest[2]  = 40;
blk_conv_info_example->start_pix_v_dest[2]  = 40;
blk_conv_info_example->size_h               = 80;
blk_conv_info_example->size_v               = 32;
gdt_mod_para_cfg_example->iflp_en           = 1;
gdt_mod_para_cfg_example->clrds0            = 4;
gdt_mod_para_cfg_example->clrds1            = 1;
trans_mod_cfg_example->ch_in                = 0;
trans_mod_cfg_example->ch_out               = 1;

e_gdt_err_t R_GDT_Colorsyn (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_colorsyn_para_cfg_example,
      p_callback*
)
```

Special Notes:

None.

4. Demo Projects

Sample code (demo project) using this driver is available on Renesas website. Search the website for the following document numbers:

1) r01an4755

2) r01an4810

5. Restrictions on using this driver

5.1 Image data size

The horizontal size and vertical size of the image data should be a multiple of 8 (bits). Horizontal size of the image should also be smaller than horizontal memory size. Image format example is shown in Figure 5-1.

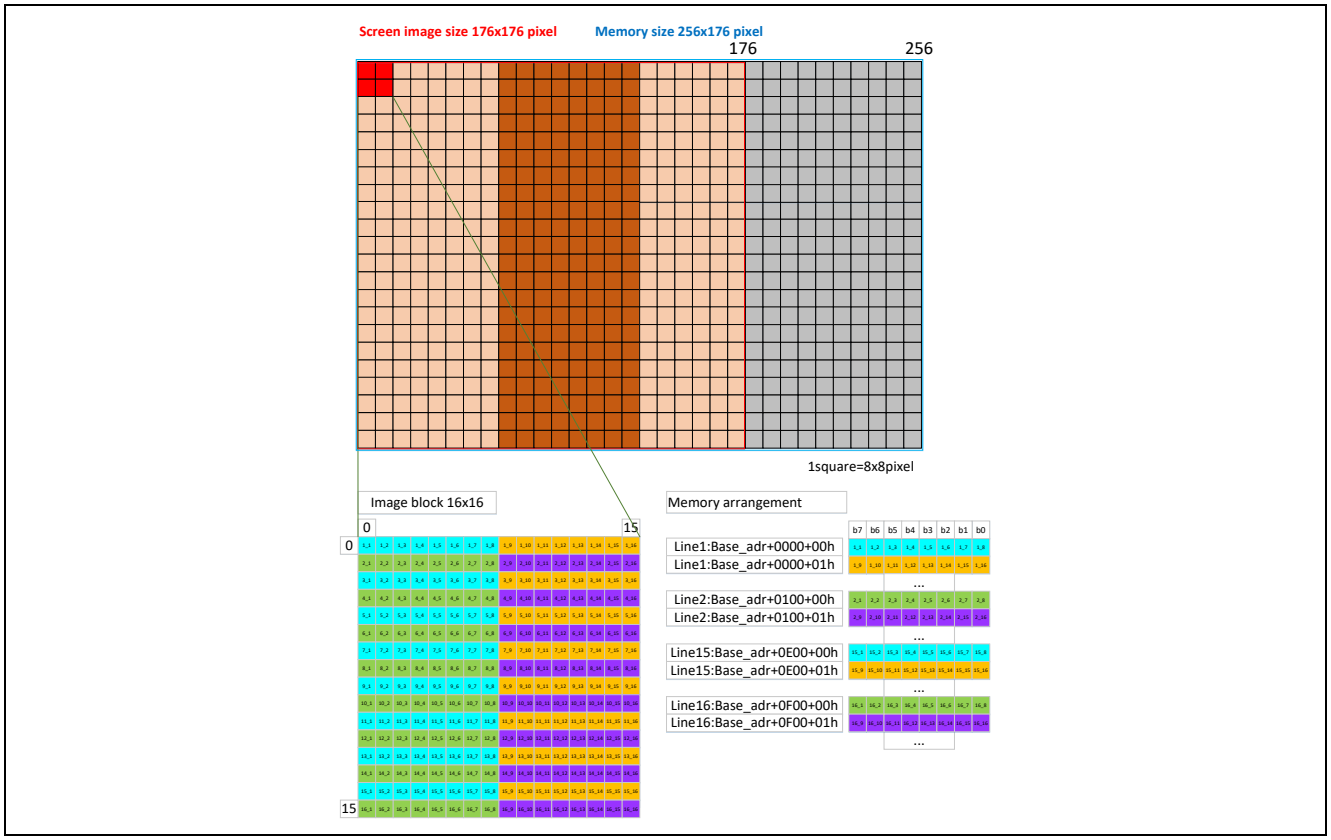

Figure 5-1 Example of image format

5.2 Horizontal memory size

When using this driver, user needs to specify the size of the work area as an argument. Select from the following sizes that are equal to or larger than the image size to be processed.

Set the horizontal memory(buffer) size to 32, 64, 128, or 256. However, set 96, 192, 384, or 768 when using the 3-bit mode of the color data alignment function, and set 128, 256, 512, or 1024 when using the 4-bit mode.

5.3 Image processing unit

For the rotation function (R_GDT_Rotate), monochrome synthesis function (R_GDT_Monochrome), and color synthesis function (R_GDT_Colorsyn), the image processing unit can be selected from two types (8 × 8 bits or 16 × 16 bits). Set the image processing data size selection variable (gdtdsz) of each function.

When using the rotation function with an image processing unit of 16 x 16 bits, set the starting address of the pre-processing and post-processing block to an even address.

The monochrome synthesis function and color synthesis function only support 8 x 8 bit image processing units. Set gdtdsz = 1 when using these functions.

5.4 DMAC interrupt settings and priority limits

This GDT driver always uses a total of 2 channels of DMAC. Set the channels that do not overlap with the DMAC channel used by other functions.

In addition, there are restrictions on the DMAC interrupt priority used by the GDT driver. The DMAC channel used by the GDT driver is set by R_GDT_DmacChSel function. This function sets “DMAC for input data transfer to GDT” and “DMAC for output data transfer”. Set the DMAC interrupt priority so that the DMAC channel selected above satisfies the following restriction.

* Limitation: Priority level of DMAC interrupts to be used

Channel used for input data transfer> Channel used for output data transfer

The interrupt priority of each channel of DMAC can be set in:

Macro definition in /Device/Config/r_dmac_cfg.h “DMACn_INT_PRIORITY (n = 0-3)”

Example: When using channel 1 for input and channel 2 for output

```
#define DMAC1_INT_PRIORITY      (0)     ///  (set from 0 to 3, 0 is highest priority.)
#define DMAC2_INT_PRIORITY      (3)     ///  (set from 0 to 3, 0 is highest priority.)
```

6. Appendices

6.1 Detailed explanation of how to use this driver

This appendix section explains the details of how to use this driver. The overview of how to use this driver is described in Section 1.3. In this section, Rotate API function is used as an example. However, this usage example also applies to other APIs.

Step 0: To prepare the data buffers.

At least three variables are needed to use GDT Driver.

- **Image data**
- **Buffer for working area**
- **Output image buffer**

GDT driver has restriction for the horizontal size. Acceptable sizes are 32, 64,128 or 256 pixels.

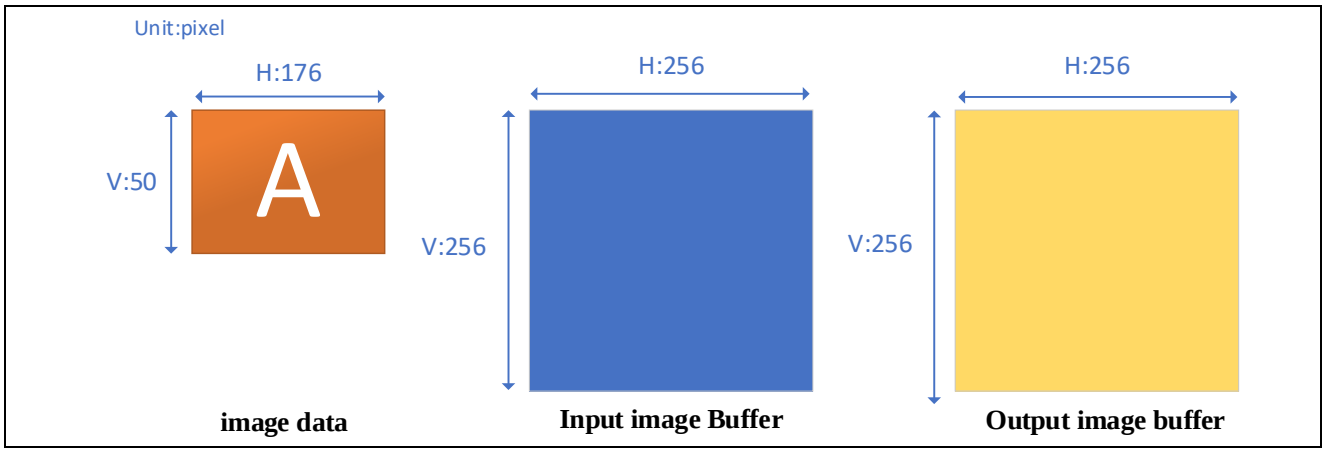

Figure 6-1 Required data area for GDT driver

Step 1: Copy Input image data to Buffer for working area

Image data needs to be copied to avoid driver restriction. This step can be skipped if horizontal size of original image data already complies to the driver acceptable size.

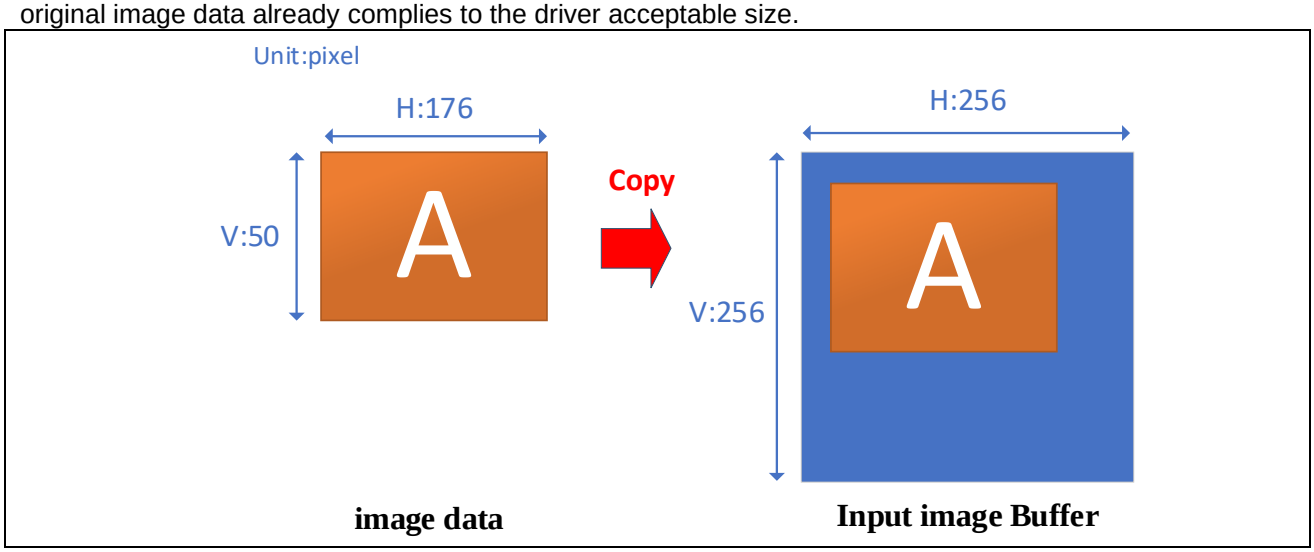

Figure 6-2 Copy the input image data

Step 2: Call the API function for GDT

This section explains the argument used for Rotate feature of GDT driver. Only 1st and 2nd argument, which corresponds to in/out data, are explained in this section. For explanation about other arguments, see Example code in Section 3.

Refer to Figure 6-4 for procedures to set the API arguments. Then, call the API.

GDT API needs roughly three information: “Display size”, “Buffer information”, and “Output image buffer”.

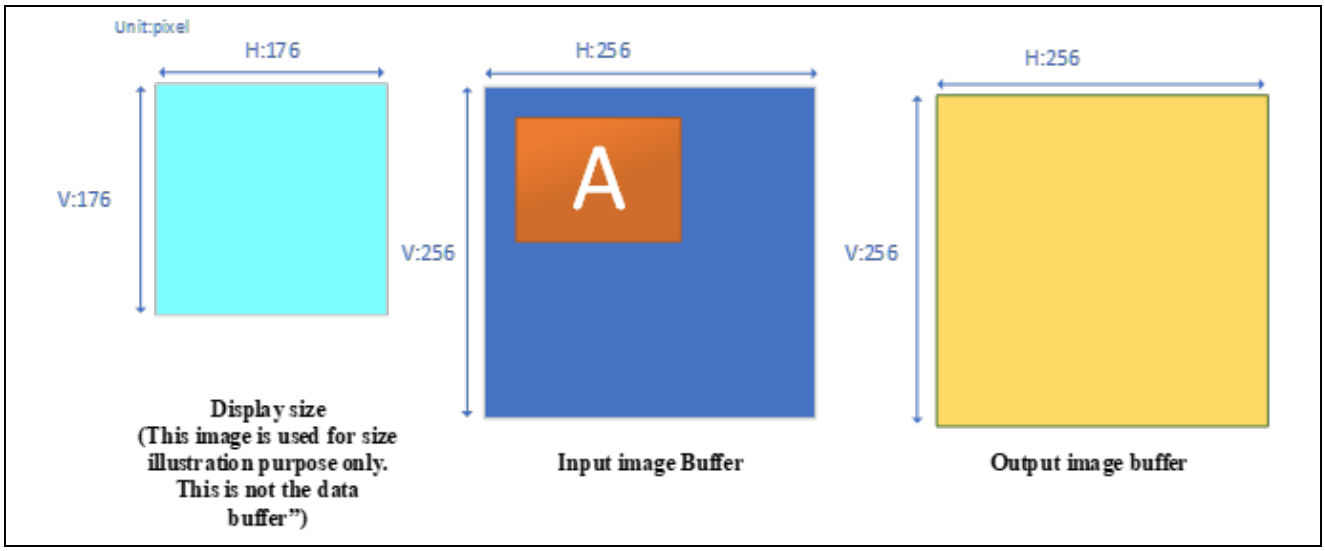

Figure 6-3 Required data for GDT API functions

The following code is quoted from API function on section 3.

***R_GDT_Rotate API Interface**

```
e_gdt_err_t     R_GDT_Rotate (
                st_img_in_info_t*               img_src,/* source image information */
                st_img_out_info_t*              img_dest,/* destination image information */
                st_blk_conv_info_t*             blk_conv_info,/* information of processing block */
                st_gdt_rotate_para_cfg_t*       gdt_rotate_para_cfg, /* GDT rotate function setting */
                gdt_cb_event_t const   p_callback     /* callback function*/
)
```

The following structures are arguments for Rotate API function. The color corresponds with the color in Figure 6-4.

***Structure detail of st_img_in_info, st_img_out_info, and st_blk_conv_info_t**

```
typedef struct
{
    uint8_t         img_num;            /*!< the number of source images */
    uint32_t        start_addr[IMG_IN_FRAME_NUM];   /*!< source image address*/
    uint32_t        size_h;             /*!< source image horizontal size*/
    uint32_t        size_v;             /*!< source image vertical size*/
    uint32_t        mem_size_h;         /*!< source image memory horizontal size */
} st_img_in_info_t;

typedef struct
{
    uint8_t         img_num;            /*!< the number of destination images      */
    uint32_t        start_addr[IMG_OUT_FRAME_NUM];  /*!< destination image address   */
    uint32_t        size_h;             /*!< destination image horizontal size  */
    uint32_t        size_v;             /*!< destination image vertical size */
    uint32_t        mem_size_h;         /*!< destination image memory horizontal size */
} st_img_out_info_t;

typedef struct
{
    uint32_t        start_pix_h_src[IMG_IN_FRAME_NUM];
          /*!< start pixel of source image, horizontal direction    */
    uint32_t        start_pix_v_src[IMG_IN_FRAME_NUM];
         /*!< start pixel of source image, vertical direction     */
    uint32_t        start_pix_h_dest[IMG_OUT_FRAME_NUM];
        /*!< start pixel of destination image, horizontal direction  */
```

```
    uint32_t        start_pix_v_dest[IMG_OUT_FRAME_NUM];
         /*!< start pixel of destination image, vertical direction */
    uint32_t        src_size_h;/*!< the size of modification area, horizontal direction
*/
    uint32_t        src_size_v;/*!< the size of modification area, vertical direction
*/
} st_blk_conv_info_t;
```

Figure 6-4 shows which information is required to provide to Figure 6-3.

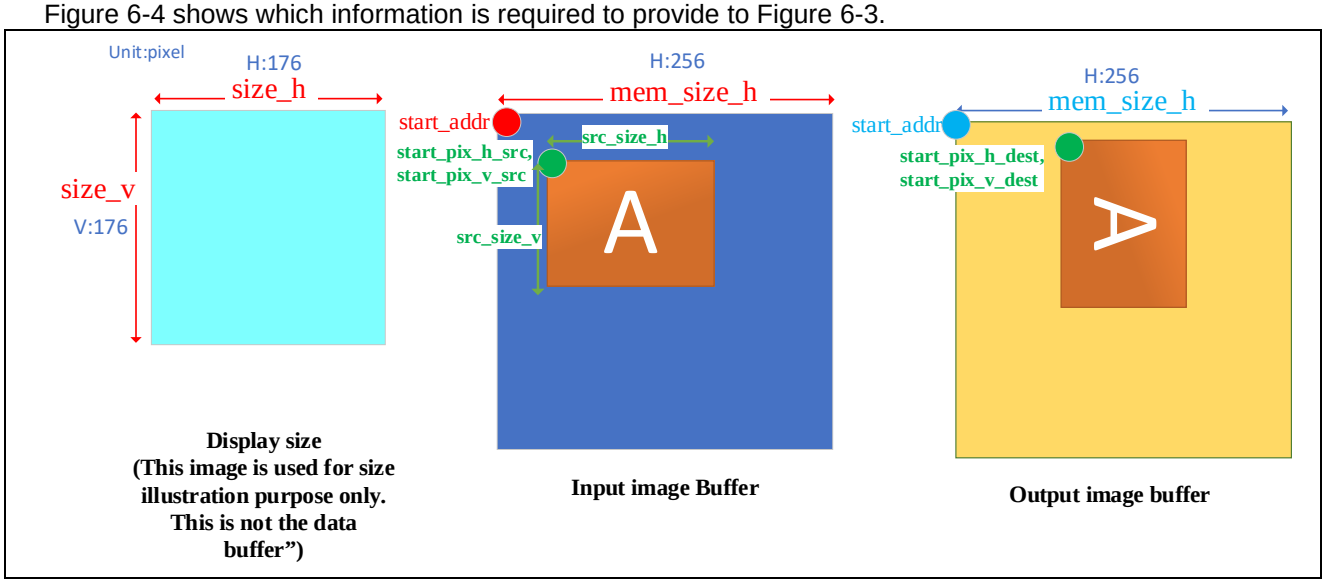

Figure 6-4 Corresponding argument for GDT API functions

7. Reference Documents

User's Manual: Hardware

RE01 1500KB Group User's Manual: Hardware R01UH0796
RE01 256KB Group User's Manual: Hardware R01UH0894
(The latest version can be downloaded from the Renesas Electronics website.)

RE01 Group CMSIS Package Getting Started Guide

RE01 1500KB, 256KB Group Getting Started Guide to Development Using CMSIS Package R01AN4660

(The latest version can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News

(The latest version can be downloaded from the Renesas Electronics website.)

User's Manual: Development Tools

(The latest version can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.15.2019	-	First edition issued
1.01	Nov.26.2019	Program Program	Fixed the RAM / ROM placement in r_gdt_cfg.h. Fixed the problem that the following two internal functions are not placed in RAM even if they are set in RAM. v_gdt_dmac_blk_upinf_in_array function e_gdt_judge_cial_dest_mem_size function Remove unused definitions from r_gdt_api.h and r_gdt_cfg.h
1.02	Dec.16.2019	- Program (256KB)	Compatible with 256KB group Modified to match 256KB IO definition
1.03	Feb.21.2020	Program	Fixed the defect of internal function v_gdt_cpuline_byte_rd_gdt_limit
1.04	Jul.10.2020	- 9 Program	Error correction Added description of definition below Replace asm("nop"); with __NOP();
1.05	Nov.05.2020	-	Error correction

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

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

Contact information

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