RE01 1500KB、256KB グループ

CMSIS ドライバ GDT 仕様書

要旨

本書では、RE01 1500KB、256KB グループ向け CMSIS software package の GDT ドライバ（以下、GDT ドライバ）の詳細仕様を説明します。本ドライバは、GDT を使用して画像処理を行います。

動作確認デバイス

RE01 1500KB グループ製品
RE01 256KB グループ製品

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 概要

1.1 ドライバ概要

GDT ドライバは、2 チャネルの DMAC を使用して、以下の画像処理を実行します。

- 縮小
- エンディアン変換
- モノクロ合成
- 回転
- スクロール
- フォント展開
- 変更なし(反転)
- カラーデータ整列
- カラー合成
- カラー化

1.2 本ドライバの使用方法

ここでは、本ドライバを使用する方法を概説します。

詳細な説明は、6.1 章に記載されています。

画像処理に関しては、以下の手順を参照してください。

ステップ 1：画像データを用意します。

1. 画像サイズ：Horizontal_size x Vertical_size

   この例では 176 * 176 ピクセルを使用しています。従って、データは uint8_t img_data[(176/8)*176] と定義されます。

2. 画像の開始アドレス：Image start address

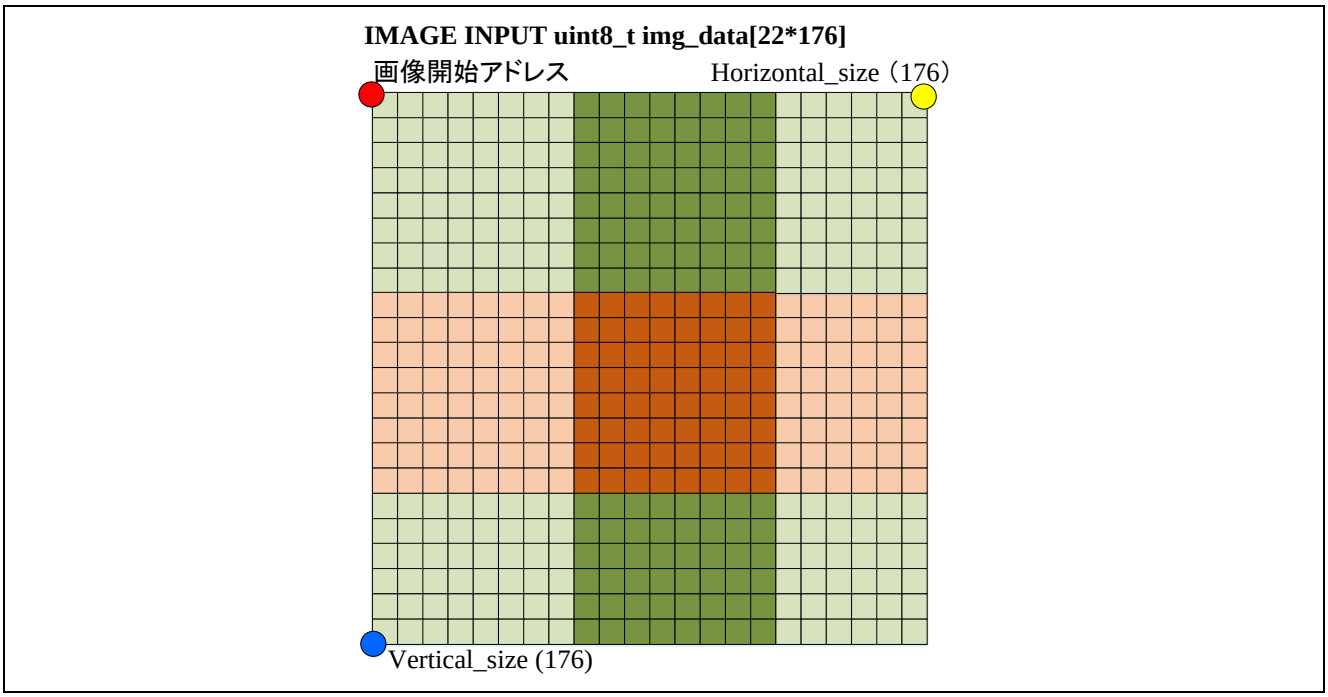

図 1-1 GDT 処理前の画像情報

ステップ 2：ステップ 1 で用意した画像を入力用のバッファメモリ IMAGE_BUF_SRC にコピーします。

本ドライバには、メモリの水平サイズに関する制約があります。水平メモリサイズは、32、64、128、256 ピクセルのいずれかである必要があります。詳細は 5.2 章を参照してください。したがって、ステップ 1 で用意した画像がこれらの水平サイズと異なる場合は、GDT ドライバを実行させる前に、データを上述のサイズのメモリ空間にコピーする必要があります。

図 1-2 は、ステップ 1 のデータを 256*176 ピクセル（32*176 バイト）のメモリにコピーした画像です。

1. 入力用メモリの水平サイズ：Memory_size
2. 画像のサイズ：Horizontal_size x Vertical_size
3. 入力用メモリの開始アドレス：&IMAGE_BUF_SRC

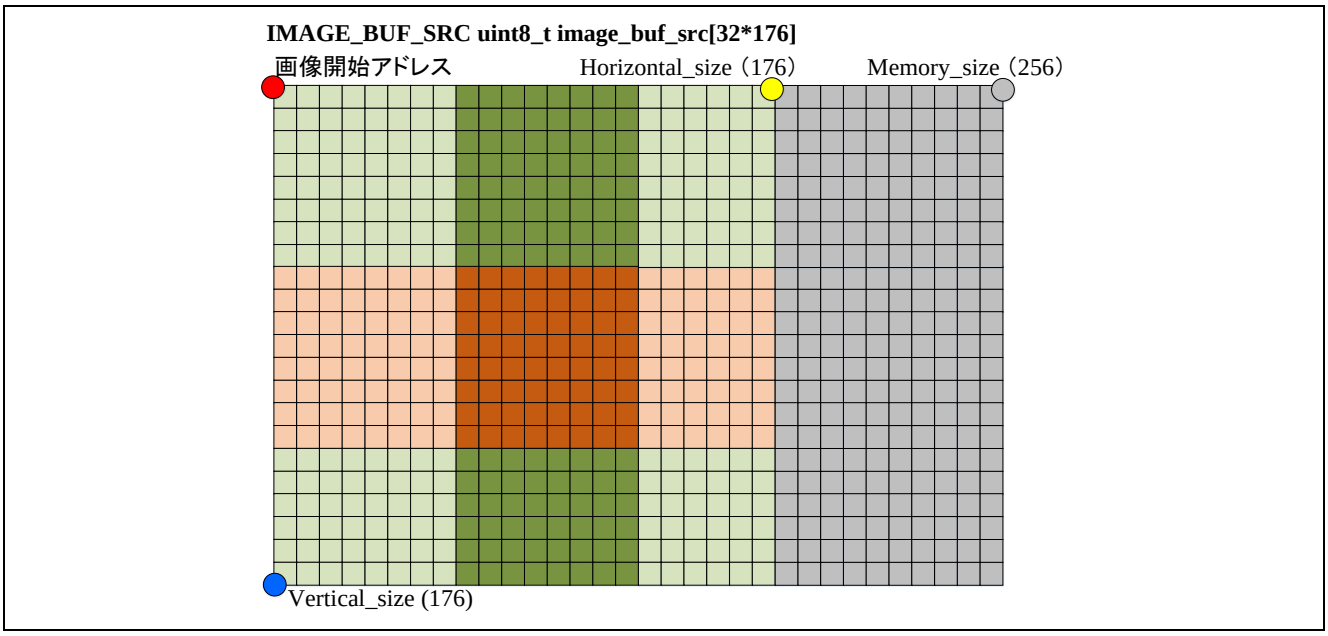

図 1-2 入力用バッファメモリの画像情報

ステップ 3： GDT ドライバを使用して画像データを変換します。

GDT 処理の前にブロック情報を設定します。
ステップ 2 で作成した入力用メモリの開始アドレスを、GDT ドライバへの入力アドレスに設定します。
また、出力データ用のバッファメモリを用意して、そのアドレスを GDT ドライバに設定する必要があります。
さらに、変換領域に関して以下を選択します。

1. 処理ブロックの開始ピクセル座標：(H,V)
2. 処理ブロックの水平サイズ：horizontal_size
3. 処理ブロックの垂直サイズ：vertical_size

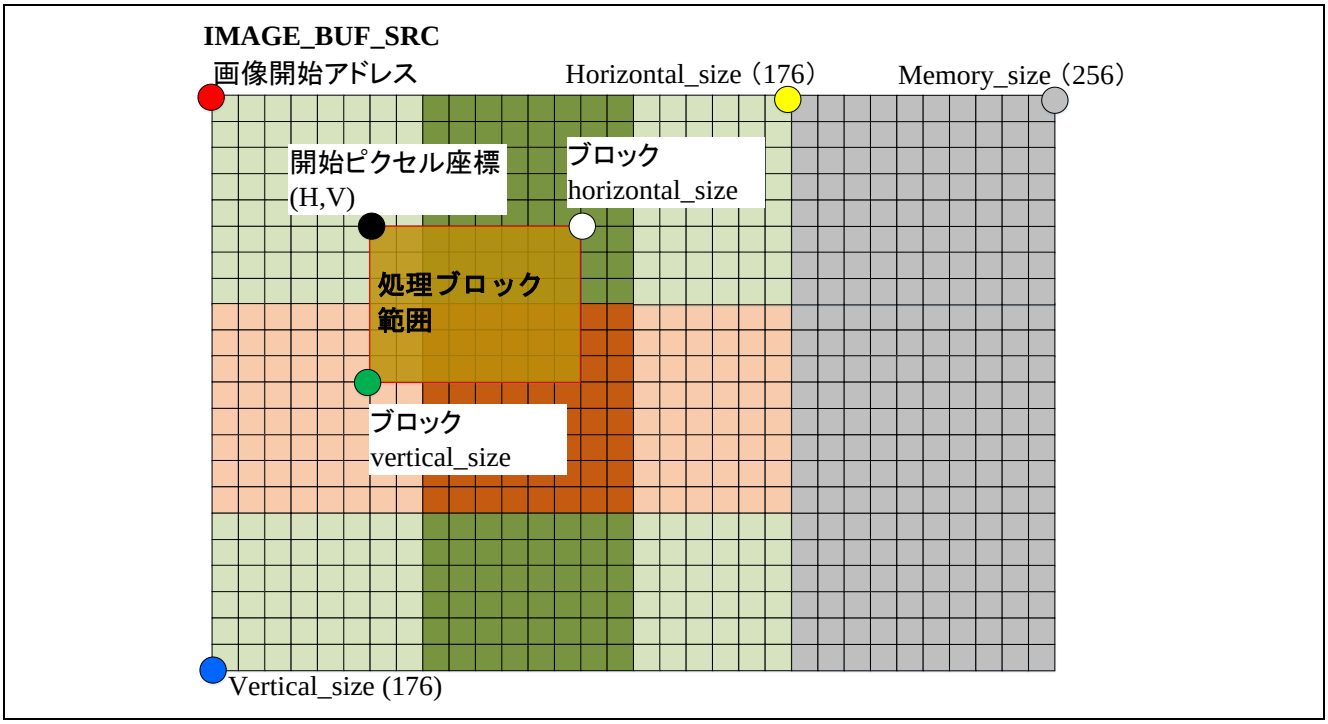

図 1-3 ブロック情報

GDT 処理後の出力画像情報とブロック情報は以下のとおりです。

1. 出力用メモリの水平サイズ：Memory_size
2. 画像サイズ：Horizontal_size x Vertical_size
3. 出力用メモリの開始アドレス：&IMAGE_BUF_DEST
4. 処理済みブロックの開始座標：(H,V)

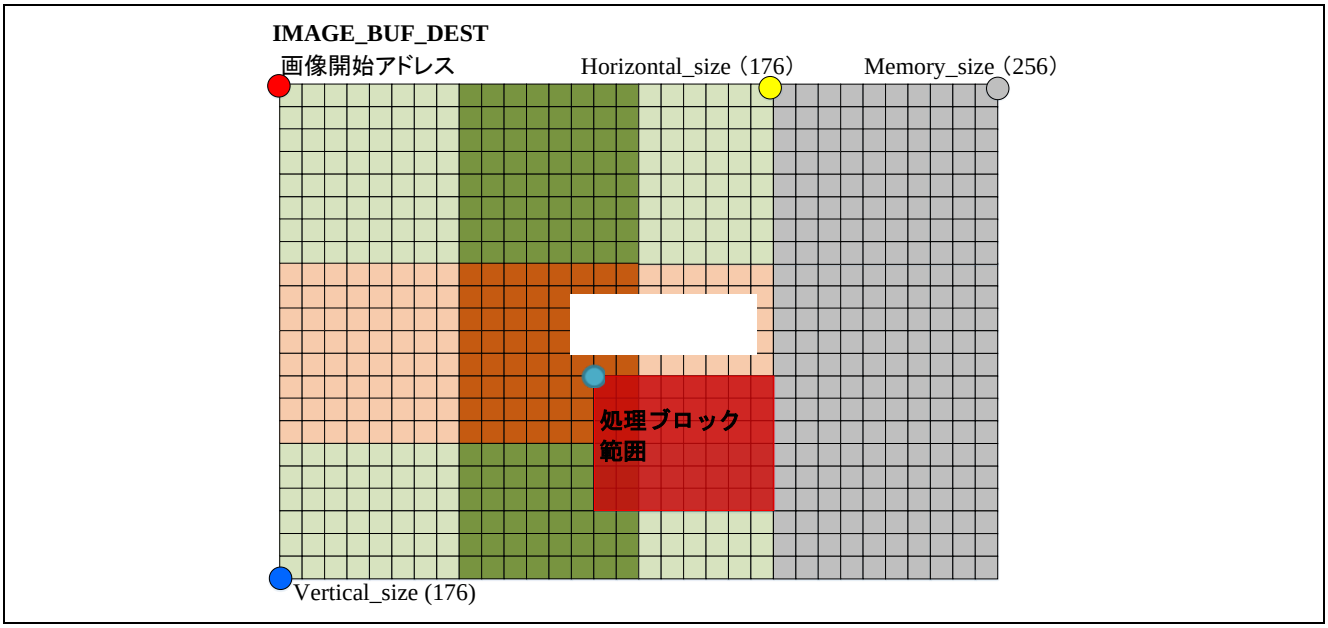

図 1-4 出力画像情報とブロック情報

1.3 API 関数一覧

表 1-1 に、本モジュールに含まれる関数を示します。

表 1-1 API 関数一覧

関数	説明
R_GDT_GetVersion	ドライバのバージョン情報を取得
R_GDT_Open	GDT のドライバを使用可能状態にします。 また、GDT H/W へのクロック供給を開始します。
R_GDT_Close	GDT のドライバを使用不可能状態にします。 また、GDT H/W へのクロック供給を停止します。
R_GDT_DmacChSel	GDT への Input と GDT からのデータ Output に使用する計 2 チャネルの DMAC チャネルを選択します。
R_GDT_Shrink	画像の縮小
R_GDT_Endian	画像のエンディアン変換
R_GDT_Monochrome	モノクロ画像の合成
R_GDT_Rotate	画像の回転
R_GDT_Scroll	画像のスクロール
R_GDT_Fount	フォント展開
R_GDT_Nochange	データ変更なし
R_GDT_Coloralign	カラーデータ整列
R_GDT_Colorsyn	カラー画像の合成
R_GDT_Color	カラー化

1.4　処理例

図 1-5 に処理の例を示します。

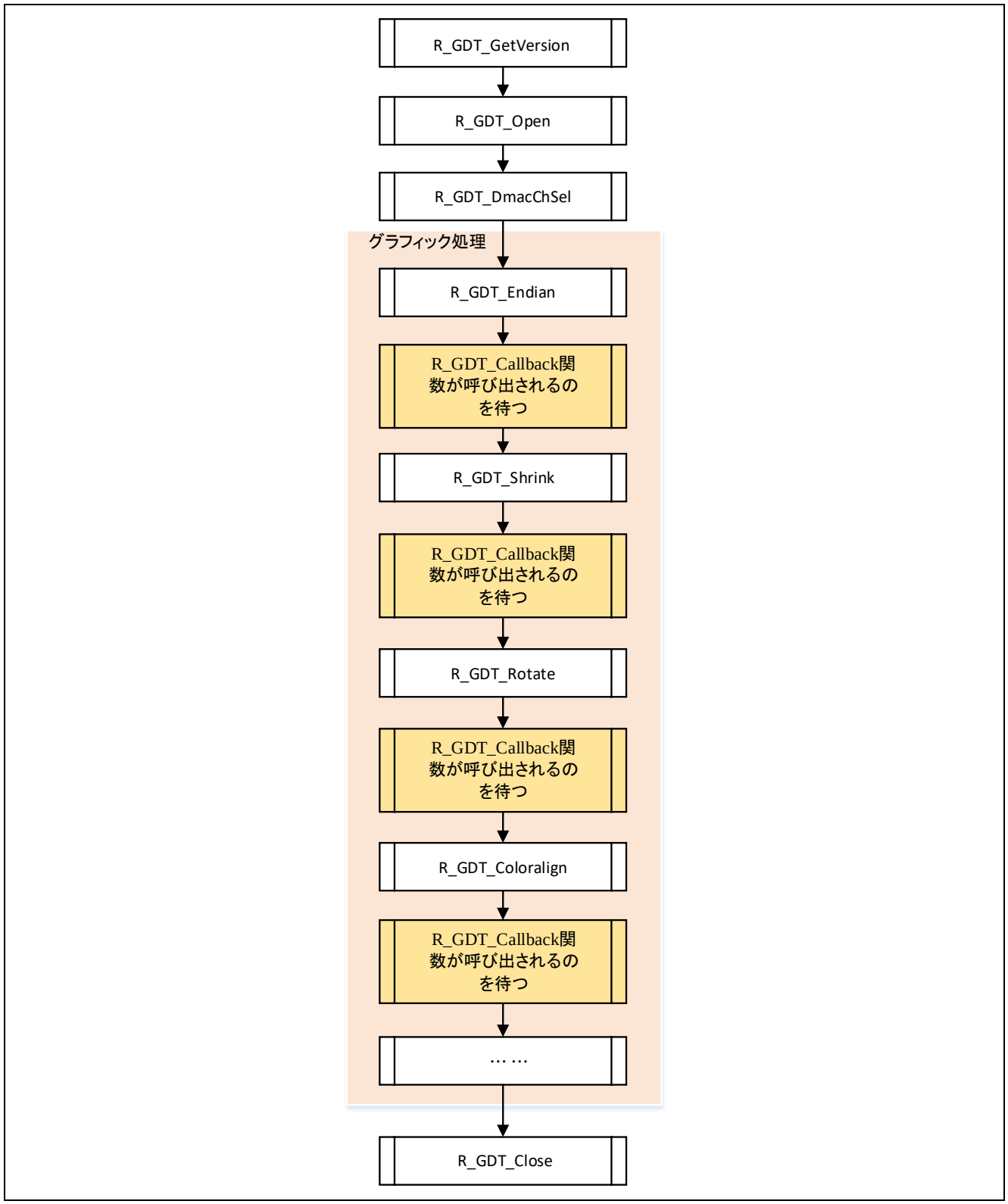

図 1-5　複数の API の呼び出し例

2. API 情報

本モジュールは以下の条件で動作することが確認されています。

2.1 ハードウェア要件

ご使用の MCU が以下の機能をサポートしている必要があります。

- DMAC(GDT へのデータ Input 用、Output 用の計 2 チャネルを使用)
- GDT

2.2 ソフトウェア要件

本ドライバは以下の CMSIS モジュールに依存しています。

- RE01 1500KB、256KB グループ向け CMSIS software package

2.3 サポートされているツールチェーン

本ドライバは以下のツールチェーンで動作することが確認済みです。

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

2.4 割り込みベクタ

GDT_DATII 割り込み、GDT_DATOI 割り込み、GDT_FDCENDI 割り込みは、
v_gdt_imgdata_trans_irq_enable()関数を実行することにより有効になります。

2 チャネルの DMAC が必要です。DMACx_INT 割り込みは、R_DMACn_InterruptEnable 関数を実行することにより有効になります。

表 2-1 に GDT ドライバ内で使用される割り込みベクタを示します。

表 2-1 GDT ドライバ内で使用される割り込みベクタ

デバイス	割り込みベクタ
RE01 1500KB RE01 256KB	**GDT の 3 つの割り込みが使用されます。** GDT_DATII 割り込み（イベント番号: A8h） GDT_DATOI 割り込み（イベント番号:A9h） GDT_ FDCENDI 割り込み（イベント番号:AAh） **合計 4 チャネルの DMAC のうち 2 チャネルを選択します。** DMAC0_INT 割り込み（イベント番号: 09h） DMAC1_INT 割り込み（イベント番号: 0Ah） DMAC2_INT 割り込み（イベント番号: 0Bh） DMAC3_INT 割り込み（イベント番号: 0Ch） 注意：DMAC の割り込み優先レベルは以下の条件を満たすよう設定してください。（r_dmac_cfg.h の割り込み定義で設定する） GDT へのデータ Input 用 DMAC チャネルの割り込み優先度 > データ Output 用 DMAC チャネルの割り込み優先度

例： 縮小処理を実施するために、DMAC0 を使用して GDT 入力データ転送し、DMAC1 を使用して GDT 出力データ転送する場合

①SYSTEM_IRQ_EVENT_NUMBER (r_system_cfg.h)の設定

```
#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)
```

②DMAC チャネルの設定

```
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);
```

表 2-2 に ICU イベントリンク設定パラメータ、表 2-3 に DMAC イベントリンク設定パラメータを示します。

表 2-2　ICU イベントリンク設定パラメータ

番号	割り込み名	レジスタ（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	GDT_FDCENDI	IELSR 5/13/21/29	5'b11110
4	DMAC0	IELSR 0/4/8/12/16/20/24/28	5'b00010
5	DMAC1	IELSR 1/5/9/13/17/21/25/29	5'b00010
6	DMAC2	IELSR 2/6/10/14/18/22/26/30	5'b00010
7	DMAC3	IELSR 3/7/11/15/19/23/27/31	5'b00010

表 2-3　DMAC イベントリンク設定パラメータ

番号	割り込みイベントリンク	レジスタ（DELSR）	DELS[7:0]	チャネル
1	GDT_DATII	DELSR0	168	DMAC0
		DELSR1		DMAC1
		DELSR2		DMAC2
		DELSR3		DMAC3
2	GDT_DATOI	DELSR0	169	DMAC0
		DELSR1		DMAC1
		DELSR2		DMAC2
		DELSR3		DMAC3
3	GDT_FDCENDI	DELSR0	170	DMAC0
		DELSR1		DMAC1

		DELSR2		DMAC2
		DELSR3		DMAC3

2.5　ヘッダファイル

すべての API 呼び出しとその対応するインターフェイス定義は r_gdt_api.h にあります。

2.6　コンフィギュレーション

本モジュールの構成オプションの設定は r_gdt_cfg.h にあります。オプション名と設定値を表 2-4 に示します。

表 2-4　構成情報

r_gdt_cfg.h の構成オプション		設定範囲	初期値
GDT_CFG_COLOR_ON	カラーデータ処理の設定	0：モノクロ 1：カラー（初期設定） メモリ節約のため、モノクロ処理のみを使用する場合は、0 に設定します。	1

2.7　コードサイズ

本モジュールに関連する代表的なコードサイズを下表に示します。表には、RE01 1500KB グループの情報を示しています。

ROM（コードと定数）サイズと RAM（グローバルデータ）サイズは、2.6 章で説明するビルドタイム設定オプションで決まります。表には、2.3 章で説明するように、C コンパイラのコンパイルオプションがデフォルト値に設定されている場合の参照値を示しています。コンパイルオプションのデフォルト値は、最適化レベル：2、最適化の種類：サイズ、データエンディアン: リトルエンディアンです。コードサイズは、C コンパイラのバージョンやコンパイルオプションによって異なります。GDT ドライバでのコードサイズを表 2-5 に示します。

表 2-5　コードサイズ

ROM、RAM、スタックコードのサイズ				
デバイス	カテゴリ	使用メモリ		備考
		全機能を RAM に割り当て	全機能を ROM に割り当て	
RE01 1500KB	ROM	4856 バイト	19856 バイト	
	RAM	140528 バイト	125416 バイト	
	スタックの最大使用サイズ	620 バイト		ネスト化された割り込みは禁じられているため、1 チャネルを使用した場合の最大値を示しています。

※コードサイズは次の条件で計測したものです。

バッファ：6 画像分使用、各画像サイズ：256x256 ビット、データ転送：2 チャネルの DMAC を使用

2.8 パラメータ

このセクションでは、本モジュールの API 関数が使用するパラメータ構造を説明します。この構造は、API のプロトタイプ宣言として r_gdt_api.h に置かれます。

```
img_src *                       /* 入力画像情報 */
        ->img_num,                                  /* 入力画像の数*/
        ->start_addr[IMG_IN_FRAME_NUM],             /* 入力画像のアドレス */
        ->size_h [IMG_IN_FRAME_NUM],                /* 入力画像の水平サイズ */
        ->size_v [IMG_IN_FRAME_NUM],                /* 入力画像の垂直サイズ */
        ->mem_size_h[IMG_IN_FRAME_NUM]  /* 入力画像メモリの水平サイズ */
img_dest*                       /* 出力画像情報 */
        ->img_num,                                  /* 出力画像の数 */
        ->start_addr[IMG_OUT_FRAME_NUM],            /* 出力画像のアドレス */
        ->size_h[IMG_OUT_FRAME_NUM],                /* 出力画像の水平サイズ */
        ->size_v[IMG_OUT_FRAME_NUM],                /* 出力画像の垂直サイズ */
        ->mem_size_h[IMG_OUT_FRAME_NUM]             /* 出力画像メモリの水平サイズ */
blk_conv_info*          /* 変換ブロック情報 */
        ->start_pix_h_src[IMG_IN_FRAME_NUM],        /* 入力画像の開始ピクセル、水平方向 */
        ->start_pix_v_src[IMG_IN_FRAME_NUM],        /* 入力画像の開始ピクセル、垂直方向 */
        ->start_pix_h_dest[IMG_OUT_FRAME_NUM],      /* 出力画像の開始ピクセル、水平方向 */
        ->start_pix_v_dest[IMG_OUT_FRAME_NUM],      /* 出力画像の開始ピクセル、垂直方向 */
        ->size_h,                                   /* 変換領域のサイズ、水平方向 */
        ->size_v                                    /* 変換領域のサイズ、垂直方向 */
gdt_shrink_para_cfg*    /* 縮小機能の設定 */
        ->iflp_en,                                  /* 反転機能有効化 */
        ->shrnk_size,                               /* 縮小サイズの選択 */
        ->shrnk_bak_grnd_colr                       /* 余剰領域のカラー設定 */

trans_mod_cfg           /* 転送モードの選択 */
        ->ch_in                                     /* 入力転送チャネルの選択 */
        ->ch_out                                    /* 出力転送チャネルの選択 */

p_callback              /* コールバック関数*/
```

2.9 戻り値

ここでは、API 関数の戻り値について説明します。ここに列挙したものは、API のプロトタイプ宣言として r_gdt_api.h に置かれます。

```
typedef     enum(
        GDT_OK,                             /* 正常動作 */
```

```
    GDT_ERROR                   /* エラー */
    GDT_ERROR_IMG_CFG,          /* 画像設定エラー */
    GDT_ERROR_BLK_OFF_BOUND,    /* ブロック領域設定エラー */
    GDT_ERROR_BLK_CFG,          /* 入力/出力画像のブロックサイズ設定エラー */
    GDT_ERROR_MOD_CFG,          /* 画像処理内容設定エラー */
    GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
    GDT_ERROR_DMAC_CH_CFG       /* DMAC チャネル設定エラー */
) gdt_err_t
```

表 2-6 戻り値

戻り値	説明	解決策
GDT_OK	GDT は正常に機能しています	-
GDT_ERROR	1. GDT のクロック開始に失敗しました	R_LPM_ModuleStart()を参照してください
	2. GDT のクロック終了に失敗しました	R_LPM_ModuleStop()を参照してください
	3. カラーAPI を使用しますが、GDT_COLOR_ON が定義されていません	r_gdt_cfg.h #define GDT_CFG_COLOR_ON (1)に変更してください
GDT_ERROR_IMG_CFG	入出力画像に対しメモリサイズが不適切です	画像のメモリサイズの設定情報を確認してください メモリサイズ制限（32,64,128,256） Coloralign 関数： 出力画像メモリサイズの制限 3 ビットモード（96,192,384,768） 4 ビットモード（128,256,512,1024）
	入出力画像の水平サイズが正しく設定されていません。	水平サイズが 8 の倍数ではありません 水平サイズが 8 未満です 水平サイズがメモリサイズを超えています 垂直サイズが 8 の倍数ではありません 垂直サイズが 8 未満です
GDT_ERROR_BLK_OFF_BOUND	入出力ブロックの開始点の座標が範囲外です	ブロック開始座標の設定情報を確認してください
	入出力ブロックの終了点の座標が範囲外です	ブロック開始座標とブロックサイズの設定情報を確認してください
GDT_ERROR_BLK_CFG	ブロックの開始点座標が 8 の倍数ではありません	クロックの設定情報を確認してください ブロックの水平サイズが 8 の倍数ではありません ブロックの水平サイズが 8 未満です
	入出力ブロックの水平サイズが正しく設定されていません	
GDT_ERROR_MOD_CFG	GDTDSZ ビットが正しく設定されていません	GDT レジスタの設定が正しいことを確認してください
	スクロール関数の ISCREN ビットが正しく設定されていません	ISCREN レジスタを 0 に設定することはできません
	R_GDT_Fount： fdltdsz の設定が 7～63 範囲外です	GDT レジスタの設定が正しいことを確認してください
	R_GDT_Fount： f dlngsz の設定が 7～64 範囲外です	
GDT_ERROR_IMG_NUM	R_GDT_Monochrome： 縁取り機能有効： 入力画像数 3、出力画像数 1	入出力画像の番号の設定が正しいことを確認してください。

	縁取り機能無効： 入力画像数 2、出力画像数 1	
	R_GDT_Coloralign： 入力画像数 3、出力画像数 1	
	R_GDT_Color： 入力画像数 1、出力画像数 1	
	R_GDT_Colorsyn： 入力画像数 6、出力画像数 3	
	上記以外の関数： 入力画像数 1、出力画像数 1	
GDT_ERROR_DMAC_CH_CFG	選択された DMAC のチャネルは使用できません	使用可能なチャネルを選択してください
	DMAC チャネル設定エラー、 設定範囲 0～3	DMAC チャネルの設定が正しいことを確認してください

2.10 コールバック関数

本ドライバでは、ユーザーが指定したコールバック関数は、GDT 処理が完了したときに呼び出されます。

GDT 変換完了後に必要なユーザ処理がある場合は、コールバック関数に必要な処理を記述してください。

```
/*コールバック関数の使用例*/
  void R_GDT_Callback(void)
  {
       /* カスタマーコード */
       GDT_end_flag = 1; /* 例：GDT 処理が完了 */
       …
  }
```

3. API 関数仕様

3.1 R_GDT_Open()

この機能は GDT 用に用意されたもので､GDT のクロックを開始します。本 API をコールした後、他の GDT API が使用可能になります。

フォーマット

e_gdt_err_t R_GDT_Open ( )

パラメータ

なし。

戻り値

```
typedef    enum(
               GDT_OK,                    /* 正常動作 */
               GDT_ERROR                  /* エラー */
               GDT_ERROR_IMG_CFG,         /* 画像設定エラー */
               GDT_ERROR_BLK_OFF_BOUND,   /* ブロック領域設定エラー */
```

```
        GDT_ERROR_BLK_CFG,          /* 入力/出力画像のブロックサイズ設定エラー */
        GDT_ERROR_MOD_CFG,          /* 画像処理内容設定エラー */
        GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
        GDT_ERROR_DMAC_CH_CFG       /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言

説明

初期化を行い、GDT H/W を起動します。

MSTPCRC26 ビットを設定し、GDT クロックを有効にします。

リエントラント

なし。

例

なし。

特記事項

なし。

3.2 R_GDT_Close ()

この関数は、GDT ドライバの使用を終了します。GDT H/W へのクロック供給を停止し、消費電力を低減します。

フォーマット

```
e_gdt_err_t    R_GDT_Close ( )
```

パラメータ

なし。

戻り値

```
typedef     enum(
        GDT_OK,                     /* 正常動作 */
        GDT_ERROR                   /* エラー */
        GDT_ERROR_IMG_CFG,          /* 画像設定エラー */
        GDT_ERROR_BLK_OFF_BOUND,    /* ブロック領域設定エラー */
        GDT_ERROR_BLK_CFG,          /* 入力/出力画像のブロックサイズ設定エラー */
        GDT_ERROR_MOD_CFG,          /* 画像処理内容設定エラー */
        GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
```

```
    GDT_ERROR_DMAC_CH_CFG        /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ
ファイル“r_gdt_api.h”でプロトタイプ宣言

説明
MSTPCRC26 ビットを設定し、GDT クロックを無効にします。

リエントラント
なし。

例
なし。

特記事項
なし。

3.3 R_GDT_GetVersion()

この関数は GDT API のバージョンを取得します。

フォーマット

```
uint32_t        R_GDT_GetVersion ( )
```

パラメータ
なし。

戻り値
バージョン

プロパティ
ファイル“r_gdt_api.h”でプロトタイプ宣言

説明
この関数は、現在インストールされている GDT ドライバのバージョンを返します。バージョン番号は上位 2 バイトがメジャーバージョン番号、下位 2 バイトがマイナーバージョン番号となるようコード化されています。例えば、バージョン 0.72 の場合、0x00000048 が返されます。

リエントラント
なし。

例
なし。

特記事項：
なし。

3.4 R_GDT_DmacChSel()

パラメータに従って、DMAC チャネルを選択します。

フォーマット

```
uint32_t   R_GDT_DmacChSel (
            st_trans_mod_cfg_t*        trans_mod_cfg        /* 転送モードの選択 */
)
```

パラメータ

```
trans_mod_cfg                /* 転送モードの選択 */
```

戻り値

```
typedef    enum(
            GDT_OK,                          /* 正常動作 */
            GDT_ERROR                        /* エラー */
            GDT_ERROR_IMG_CFG,               /* 画像設定エラー */
            GDT_ERROR_BLK_OFF_BOUND,         /* ブロック領域設定エラー */
            GDT_ERROR_BLK_CFG,               /* 入力/出力画像のブロックサイズ設定エラー */
            GDT_ERROR_MOD_CFG,               /* 画像処理内容設定エラー */
            GDT_ERROR_IMG_NUM,               /* 画像数設定エラー */
            GDT_ERROR_DMAC_CH_CFG            /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言

説明

パラメータに従って DMAC チャネルを選択します。チャネルによってデータが GDTH/W に転送されます。

- パラメータをチェック
- DMAC チャネルを選択
- DMAC チャネルの使用可否をチェック

リエントラント

なし。

例

なし。

特記事項

GDT に使用する DMAC の割り込みレベル設定に制限があります。5.4 章をご参照ください。

3.5 R_GDT_Shrink ()

この関数は、画像を縮小する場合に呼び出され、画像を縮小します。

フォーマット

```
e_gdt_err_t   R_GDT_Shrink (
              st_img_in_info_t*          img_src,          /* 入力画像情報 */
              st_img_out_info_t*         img_dest,         /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,    /* 変換ブロック情報 */
              st_gdt_shrink_para_cfg_t*  gdt_shrink_para_cfg, /* 縮小機能の設定 */
              gdt_cb_event_t const       p_callback        /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_shrink_para_cfg*        /* 縮小機能の設定 */
        ->iflp_en               /* 反転機能有効化 */
        ->shrnk_size            /* 縮小サイズの選択 */
        ->shrnk_bak_grnd_colr   /*余剰領域のカラー設定 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef   enum(
            GDT_OK,                     /* 正常動作 */
            GDT_ERROR                   /* エラー */
            GDT_ERROR_IMG_CFG,          /* 画像設定エラー */
            GDT_ERROR_BLK_OFF_BOUND,    /* ブロック領域設定エラー */
            GDT_ERROR_BLK_CFG,          /* 入力/出力画像のブロックサイズ設定エラー */
            GDT_ERROR_MOD_CFG,          /* 画像処理内容設定エラー */
            GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
            GDT_ERROR_DMAC_CH_CFG       /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報と縮小モード設定に従って画像の選択領域が縮小し、目的の位置に出力されます。縮小比は 1/8～7/8 です。

縮小設定：

iflp_en: 0/1

shrnk_size: 0~7

shrnk_bak_grnd_colr: 0/1

本関数の処理フローについては、以下の手順を参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタを縮小モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化
- 画像処理の完了
- コールバック関数の呼び出し

リエントラント

なし。

処理例の概要

モード設定：縮小比 7/8、反転機能 ON、余剰背景色設定 0

縮小処理については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(24,40)で、処理ブロックサイズ（水平サイズ 152、垂直サイズ 32）によって、縮小処理が行われます。

開始ピクセルが(24, 40)の出力画像に保存されます。

コード例

```
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 縮小比 */
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*
)
```

特記事項

なし。

3.6 R_GDT_Endian()

この関数は画像のエンディアン変換を行います。画像のエンディアン変換を行う場合に呼び出されます。

フォーマット

```
e_gdt_err_t   R_GDT_Endian (
              st_img_in_info_t*          img_src,              /* 入力画像情報 */
              st_img_out_info_t*         img_dest,             /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,        /* 変換ブロック情報 */
              st_gdt_endian_para_cfg_t*  gdt_endian_para_cfg,  /* エンディアン変換機能の設定 */
              gdt_cb_event_t const       p_callback            /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_endian_para_cfg*        /* エンディアン変換機能の設定 */
      ->iflp_en                   /* 反転機能有効化 */
     ->endian                     /* エンディアン変換機能有効化 */
```

```
p_callback                    /* コールバック関数*/
```

戻り値

```
typedef    enum(
                GDT_OK,                          /* 正常動作 */
                GDT_ERROR                        /* エラー */
                GDT_ERROR_IMG_CFG,               /* 画像設定エラー */
                GDT_ERROR_BLK_OFF_BOUND,         /* ブロック領域設定エラー */
                GDT_ERROR_BLK_CFG,               /* 入力/出力画像のブロックサイズ設定エラー */
                GDT_ERROR_MOD_CFG,               /* 画像処理内容設定エラー */
                GDT_ERROR_IMG_NUM,               /* 画像数設定エラー */
                GDT_ERROR_DMAC_CH_CFG            /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とエンディアン変換モード設定に従って画像の選択領域がエンディアン変換処理され、目的の位置に出力されます。

エンディアン変換機能のデータ処理単位は、16x16 ビットです。

エンディアン変換設定：

iflp_en: 0/1

endian：0/1

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタをエンディアン変換モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：エンディアン変換 ON、反転機能 ON

エンディアン変換処理については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(24,40)で、処理ブロックサイズ（水平サイズ 80、垂直サイズ 32）によって、エンディアン変換処理が行われます。

開始ピクセルが(24, 40)の出力画像に保存されます。

コード例

```
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_tR_GDT_endian (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_endian_para_cfg_example,
      p_callback*
)
```

特記事項

なし。

3.7 R_GDT_Nochange()

この関数は画像を反転させます。画像を反転する場合に呼び出されます。

フォーマット

```
e_gdt_err_t    R_GDT_Nochange (
               st_img_in_info_t*        img_src,        /* 入力画像情報 */
```

```
                st_img_out_info_t*          img_dest,           /* 出力画像情報 */
                st_blk_conv_info_t*         blk_conv_info,      /* 変換ブロック情報 */
                st_gdt_nochange_para_cfg_t* gdt_nochange_para_cfg, /* 変更なし機能の設定 */
                gdt_cb_event_t const        p_callback          /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_nochange_para_cfg*      /* 変更なし機能の設定 */
    ->iflp_en                   /* 反転機能有効化 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef     enum(
                GDT_OK,                     /* 正常動作 */
                GDT_ERROR                   /* エラー */
                GDT_ERROR_IMG_CFG,          /* 画像設定エラー */
                GDT_ERROR_BLK_OFF_BOUND,    /* ブロック領域設定エラー */
                GDT_ERROR_BLK_CFG,          /* 入力/出力画像のブロックサイズ設定エラー */
                GDT_ERROR_MOD_CFG,          /* 画像処理内容設定エラー */
                GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
                GDT_ERROR_DMAC_CH_CFG       /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

R_GDT_Nochange 関数は、R_GDT_Endian 関数を呼び出します。入力画像情報と変更なしモード設定に従って画像の選択領域が反転処理され、目的の位置に出力されます。

エンディアン変換機能のデータ処理単位は、16x16 ビットです。

変更なし機能設定：
iflp_en: 0/1

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算

- レジスタを nochange モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント
なし。

例
画像サイズの詳細は、R_GDT_Endian()関数を参照してください。

処理例の概要

モード設定：エンディアン OFF、反転機能 ON

変更なし(反転処理)については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(24,40)で、処理ブロックサイズ（水平サイズ 80、垂直サイズ 32）によって、nochange 処理が行われます。

開始ピクセルが(24, 40)の出力画像に保存されます。

コード例
```
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_tR_GDT_Nochange (
```

```
        img_src_example,
        img_dest_example,
        blk_conv_info_example,
        gdt_nochange_para_cfg_example,
        p_callback*
)
```

特記事項

なし。

3.8 R_GDT_Monochrome()

この関数は、画像をモノクロ合成します。モノクロ合成を行う際に呼び出されます。

フォーマット

```
e_gdt_err_t    R_GDT_Monochrome (
               st_img_in_info_t*                img_src,        /* 入力画像情報 */
               st_img_out_info_t*               img_dest,       /* 出力画像情報 */
               st_blk_conv_info_t*              blk_conv_info,  /* 変換ブロック情報 */
               st_gdt_monochrome_para_cfg_t* gdt_monochrome_para_cfg,
                                                                /* モノクロ合成機能の設定 */
               gdt_cb_event_t const             p_callback      /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_monochrome_para_cfg*    /* モノクロ合成機能の設定 */
      ->iflp_en                 /* 反転機能有効化 */
      ->gdtdsz                  /* 画像処理データサイズ選択 0：16*16 ビット 1：8*8 ビット */
      ->mpcs                    /* モノクロ優先色指定 */
      ->mbrden                  /* 画像縁取り有効化 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef    enum(
               GDT_OK,                      /* 正常動作 */
               GDT_ERROR                    /* エラー */
               GDT_ERROR_IMG_CFG,           /* 画像設定エラー */
               GDT_ERROR_BLK_OFF_BOUND,     /* ブロック領域設定エラー */
               GDT_ERROR_BLK_CFG,           /* 入力/出力画像のブロックサイズ設定エラー */
               GDT_ERROR_MOD_CFG,           /* 画像処理内容設定エラー */
```

```
    GDT_ERROR_IMG_NUM,          /* 画像数設定エラー */
    GDT_ERROR_DMAC_CH_CFG       /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とモノクロ合成モード設定に従って画像の選択領域がモノクロ合成され、目的の位置に出力されます。

モノクロ合成関数のデータ処理単位は、8x8 ビットです。

モノクロ合成設定：

iflp_en: 0/1

gdtdsz: 0/1

mpcs: 0/1

mbrden: 0/1

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタをモノクロ合成モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：反転機能 ON、1 合成、縁取り機能有効、画像処理のデータサイズ 8x8 ビット。

モノクロ合成処理については、入力画像が 3 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC[i]のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(24,40)、(48, 40)、(120, 32)で、処理ブロックサイズ（水平サイズ 64、垂直サイズ 32）によって、モノクロ合成処理が行われます。開始ピクセル(64, 40)の出力画像に保存されます。

コード例

```
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_tR_GDT_Monochrome (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
      gdt_monochrome_para_cfg_example,
      trans_mod_cfg_example
      p_callback*
)
```

特記事項

画像処理データサイズは、8x8 ビットのみです。本 API 関数は、16x16 ビット未対応です。

3.9 R_GDT_Rotate()

この関数は、画像を回転させます。画像を回転する場合に呼び出されます。

フォーマット

```
e_gdt_err_t   R_GDT_Rotate (
              st_img_in_info_t*          img_src,          /* 入力画像情報 */
              st_img_out_info_t*         img_dest,         /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,    /* 変換ブロック情報 */
              st_gdt_rotate_para_cfg_t*  gdt_rotate_para_cfg,  /* 回転機能の設定 */
```

```
        gdt_cb_event_t const        p_callback          /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_rotate_para_cfg*        /* 回転機能の設定 */
    ->iflp_en                   /* 反転機能有効化 */
    ->gdtdsz                    /* 画像処理データサイズ選択 0：16*16 ビット 1：8*8 ビット */
      ->rttfc                   /* 回転関数選択 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef  enum(
        GDT_OK,                         /* 正常動作 */
        GDT_ERROR                       /* エラー */
        GDT_ERROR_IMG_CFG,              /* 画像設定エラー */
        GDT_ERROR_BLK_OFF_BOUND,        /* ブロック領域設定エラー */
        GDT_ERROR_BLK_CFG,              /* 入力/出力画像のブロックサイズ設定エラー */
        GDT_ERROR_MOD_CFG,              /* 画像処理内容設定エラー */
        GDT_ERROR_IMG_NUM,              /* 画像数設定エラー */
        GDT_ERROR_DMAC_CH_CFG           /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報と回転モード設定に従って画像の選択領域が回転処理され、目的の位置に出力されます。

回転関数のデータ処理単位は、8x8 ビットまたは 16x16 ビットです。

回転設定：

iflp_en: 0/1

gdtdsz: 0/1

rttfc: 0~3

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算

- レジスタを回転モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：右回転 90°、反転機能 ON、画像処理データサイズ 16x16bit.

回転処理については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(48,24)で、処理ブロックサイズ（水平サイズ 32、垂直サイズ 64）によって、回転処理が行われます。

開始ピクセルが(48, 24)の出力画像に保存されます。

コード例

```
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_tR_GDT_Rotate (
    img_src_example,
    img_dest_example,
```

```
    blk_conv_info_example,
    gdt_rotate_para_cfg_example,
    p_callback*
)
```

特記事項

なし。

3.10 R_GDT_Scroll()

この関数は画像をスクロールします。画像をスクロールする場合に呼び出されます。

フォーマット

```
e_gdt_err_t   R_GDT_Scroll (
              st_img_in_info_t*          img_src,          /* 入力画像情報 */
              st_img_out_info_t*         img_dest,         /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,    /* 変換ブロック情報 */
              st_gdt_scroll_para_cfg_t*  gdt_scroll_para_cfg, /* スクロール機能の設定 */
              gdt_cb_event_t const       p_callback        /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_scroll_para_cfg*        /* スクロール機能の設定 */
    ->iflp_en                   /* 反転機能有効化 */
    ->iscren                    /* スクロール関数有効化 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef    enum(
               GDT_OK,                          /* 正常動作 */
               GDT_ERROR                        /* エラー */
               GDT_ERROR_IMG_CFG,               /* 画像設定エラー */
               GDT_ERROR_BLK_OFF_BOUND,         /* ブロック領域設定エラー */
               GDT_ERROR_BLK_CFG,               /* 入力/出力画像のブロックサイズ設定エラー */
               GDT_ERROR_MOD_CFG,               /* 画像処理内容設定エラー */
               GDT_ERROR_IMG_NUM,               /* 画像数設定エラー */
               GDT_ERROR_DMAC_CH_CFG            /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とスクロールモード設定に従って画像の選択領域がスクロール処理され、目的の位置に出力されます。

スクロール関数のデータ処理単位は、16x16 ビットです。

スクロール設定：
iflp_en: 0/1

iscren: 1~7 (0：スクロール関数無効化)

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタをスクロールモードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：スクロール幅 1 ビット、反転機能 ON

スクロール処理については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(40,32)で、処理ブロックサイズ（水平サイズ 128、垂直サイズ 16）によって、スクロール処理が行われます。

開始ピクセルが(40, 32)の出力画像に保存されます。

コード例

```
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_tR_GDT_Scroll (
       img_src_example,
       img_dest_example,
       blk_conv_info_example,
       gdt_scroll_para_cfg_example,
       p_callback*
)
```

特記事項

なし。

3.11 R_GDT_Fount()

この関数は、フォントデータを展開します。フォントデータを展開する場合にこの関数が呼び出されます。

フォーマット

```
e_gdt_err_t   R_GDT_Fount (
              st_img_in_info_t*          img_src,            /* 入力画像情報 */
              st_img_out_info_t*         img_dest,           /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,      /* 変換ブロック情報 */
              st_gdt_fount_para_cfg_t*   gdt_fount_para_cfg, /* フォント展開機能の設定 */
              gdt_cb_event_t const       p_callback          /* コールバック関数*/
)
```

パラメータ

```
img_src *                 /* 入力画像情報 */
img_dest*                 /* 出力画像情報 */
blk_conv_info*            /* 変換ブロック情報 */
gdt_fount_para_cfg*       /* フォント展開機能の設定 */
```

```
        ->iflp_en                   /* 反転機能有効化 */
        ->sac                       /* 開始アドレス変更ビット */
        ->fdhad                     /* 空白ビットの設定 */
        ->fdltdsz                   /* フォントの水平ビット数の設定 */
        ->iscren                    /* フォントの垂直ビット数の設定 */
  p_callback                /* コールバック関数*/
```

戻り値

```
  typedef     enum(
                GDT_OK,                         /* 正常動作 */
                GDT_ERROR                       /* エラー */
                GDT_ERROR_IMG_CFG,              /* 画像設定エラー */
                GDT_ERROR_BLK_OFF_BOUND,        /* ブロック領域設定エラー */
                GDT_ERROR_BLK_CFG,              /* 入力/出力画像のブロックサイズ設定エラー */
                GDT_ERROR_MOD_CFG,              /* 画像処理内容設定エラー */
                GDT_ERROR_IMG_NUM,              /* 画像数設定エラー */
                GDT_ERROR_DMAC_CH_CFG           /* DMAC チャネル設定エラー */
  ) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とフォント展開モード設定に従って、RAM に保存されているフォントデータが目的の画像の指定位置に展開されます。

フォント展開設定：

iflp_en: 0/1

sac: 0~7

fdhad: 0/1

fdltdsz: 7~63

fdlngsz: 7/64

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタをフォント展開モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：反転機能 ON、開始アドレス変更 4 ビット

空白部分の挿入値は 0 です。

フォントサイズは H10xV10 です。

フォント展開処理については、入力画像が 1 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H192xV128、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、処理済みブロックのブロック開始座標は(0,0)で、処理ブロックサイズ（水平サイズ 0、垂直サイズ 0）によって、フォント展開処理が行われます。

開始ピクセルが(8, 2)の出力画像に保存されます。

コード例

```
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_tR_GDT_Fount (
      img_src_example,
      img_dest_example,
      blk_conv_info_example,
```

```
        gdt_fount_para_cfg_example,
        p_callback*
)
```

特記事項

なし。

3.12 R_GDT_Coloralign()

この関数はカラーデータを整列します。カラーデータを整列する場合に呼び出されます。

フォーマット

```
e_gdt_err_t    R_GDT_Coloralign (
               st_img_in_info_t*              img_src,              /* 入力画像情報 */
               st_img_out_info_t*             img_dest,             /* 出力画像情報 */
               st_blk_conv_info_t*            blk_conv_info,        /* 変換ブロック情報 */
               st_gdt_coloralign_para_cfg_t*  gdt_coloralign_para_cfg, /* カラーデータ整列機能の設定 */
               gdt_cb_event_t const           p_callback            /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_coloralign_para_cfg*    /* カラーデータ整列機能の設定 */
    ->iflp_en                   /* 反転機能有効化 */
    ->cialgsl                   /* カラーデータ整列モードの選択 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef    enum(
               GDT_OK,                      /* 正常動作 */
               GDT_ERROR                    /* エラー */
               GDT_ERROR_IMG_CFG,           /* 画像設定エラー */
               GDT_ERROR_BLK_OFF_BOUND,     /* ブロック領域設定エラー */
               GDT_ERROR_BLK_CFG,           /* 入力/出力画像のブロックサイズ設定エラー */
               GDT_ERROR_MOD_CFG,           /* 画像処理内容設定エラー */
               GDT_ERROR_IMG_NUM,           /* 画像数設定エラー */
               GDT_ERROR_DMAC_CH_CFG        /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とカラーデータ整列モード設定に従って R 画像データ、G 画像データ、および G 画像データが整列され、目的の位置に出力されます。

カラーデータ整列のデータ処理単位は、16x16 ビットです。

カラーデータ整列設定：

iflp_en: 0/1

cialgsl: 0/1

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定
- 転送情報を計算
- レジスタをカラーデータ整列モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：反転機能 ON、カラーデータ整列は 3 ビットデータモード

エンディアン変換処理については、入力画像が 3 つ、出力画像が 1 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：769

処理ブロック設定：

入力画像では、R ブロック、G ブロック、B ブロックのブロック開始座標は(H16, V16) (H24, V40) (H80, V8)で、その処理ブロックサイズ(horizontal_size 32, vertical_size 32)が、カラーデータ整列処理を行います。

開始ピクセルが(48, 56)の出力画像に保存されます。

コード例

```
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_tR_GDT_Coloralign (
            img_src_example,
            img_dest_example,
            blk_conv_info_example,
            gdt_coloralign_para_cfg_example,
            p_callback*
)
```

特記事項

なし。

3.13 R_GDT_Colorsyn()

この関数はカラーデータを合成します。カラーデータを合成する場合に呼び出されます。

フォーマット

```
e_gdt_err_t   R_GDT_Colosyn (
              st_img_in_info_t*          img_src,              /* 入力画像情報 */
              st_img_out_info_t*         img_dest,             /* 出力画像情報 */
              st_blk_conv_info_t*        blk_conv_info,        /* 変換ブロック情報 */
              st_gdt_colorsyn_para_cfg_t* gdt_colorsyn_para_cfg, /* カラー合成機能の設定 */
```

```
                gdt_cb_event_t const        p_callback          /* コールバック関数*/
)
```

パラメータ

```
img_src *                   /* 入力画像情報 */
img_dest*                   /* 出力画像情報 */
blk_conv_info*              /* 変換ブロック情報 */
gdt_colorsyn_para_cfg*      /* カラー合成機能の設定 */
    ->iflp_en                   /* 反転機能有効化 */
    ->gdtdsz                    /* 画像処理データサイズ選択 0：16*16 ビット 1：8*8 ビット */
    ->cpts                      /* 優先・透過設定 */
    ->cdcs                      /* 指定色設定 */
p_callback                  /* コールバック関数*/
```

戻り値

```
typedef    enum(
            GDT_OK,                         /* 正常動作 */
            GDT_ERROR                       /* エラー */
            GDT_ERROR_IMG_CFG,              /* 画像設定エラー */
            GDT_ERROR_BLK_OFF_BOUND,        /* ブロック領域設定エラー */
            GDT_ERROR_BLK_CFG,              /* 入力/出力画像のブロックサイズ設定エラー */
            GDT_ERROR_MOD_CFG,              /* 画像処理内容設定エラー */
            GDT_ERROR_IMG_NUM,              /* 画像数設定エラー */
            GDT_ERROR_DMAC_CH_CFG           /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とカラー合成モード設定に従って画像の選択領域が合成され、目的の位置に出力されます。

カラー合成設定：

iflp_en: 0/1

gdtdsz:0/1

cpts: 0/1

cdcs:0~7

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- GDT 割り込みの優先度を設定

- 転送情報を計算
- レジスタをカラー合成モードに設定
- 転送シーケンスを開始
- GDT 割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：反転機能 ON、画像処理データサイズ 16x16 ビット

優先色モード、指定色は赤

エンディアン変換処理については、入力画像が 6 つ、出力画像が 3 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC[i]のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST[j]のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、前景の R ブロック、前景の G ブロック、前景の B ブロック、後景の R ブロック、後景の G ブロック、後景の B ブロックのブロック開始座標は、(H16, V16) (H16, V16) (H16, V16) (H32, V40) (H32, V40) (H32, V40)で、その処理ブロックサイズ(horizontal_size 128, vertical_size 64)がカラー合成処理を行います。

開始ピクセルが(48, 64)の出力画像に保存されます。

コード例

```
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_tR_GDT_Colorsyn (
       img_src_example,
       img_dest_example,
       blk_conv_info_example,
       gdt_colorsyn_para_cfg_example,
       p_callback*
)
```

特記事項

なし。

3.14 R_GDT_Color()

この関数はモノクロ画像をカラー画像に変換します。

フォーマット

```
e_gdt_err_t    R_GDT_Color (
               st_img_in_info_t*          img_src,              /* 入力画像情報 */
               st_img_out_info_t*         img_dest,             /* 出力画像情報 */
               st_blk_conv_info_t*        blk_conv_info,        /* 変換ブロック情報 */
               st_gdt_color_para_cfg_t*   gdt_color_para_cfg,   /* カラー化機能の設定 */
               gdt_cb_event_t const       p_callback            /* コールバック関数*/
)
```

パラメータ

```
img_src *                  /* 入力画像情報 */
img_dest*                  /* 出力画像情報 */
blk_conv_info*             /* 変換ブロック情報 */
gdt_color_para_cfg*        /* カラー化機能の設定 */
    ->iflp_en                  /* 反転機能有効化 */
    ->clrds0                   /* 0 データのカラー設定 */
    ->clrds1                   /* 1 データのカラー設定 */
p_callback                 /* コールバック関数*/
```

戻り値

```
typedef    enum(
               GDT_OK,                      /* 正常動作 */
               GDT_ERROR                    /* エラー */
               GDT_ERROR_IMG_CFG,           /* 画像設定エラー */
               GDT_ERROR_BLK_OFF_BOUND,     /* ブロック領域設定エラー */
               GDT_ERROR_BLK_CFG,           /* 入力/出力画像のブロックサイズ設定エラー */
               GDT_ERROR_MOD_CFG,           /* 画像処理内容設定エラー */
               GDT_ERROR_IMG_NUM,           /* 画像数設定エラー */
               GDT_ERROR_DMAC_CH_CFG        /* DMAC チャネル設定エラー */
) gdt_err_t
```

プロパティ

ファイル“r_gdt_api.h”でプロトタイプ宣言。

説明

入力画像情報とカラー化モード設定に従って画像の選択領域がカラー化され、目的の位置に出力されます。

カラー化設定：
iflp_en: 0/1

clrds0: 0~7

clrds1:0~7

本関数の処理フローは次のステップを参照してください。

- パラメータをチェック
- 本関数で使用する変数を初期化
- ＧＤＴ割り込みの優先度を設定
- 転送情報を計算
- レジスタをカラー化モードに設定
- 転送シーケンスを開始
- ＧＤＴ割り込みを有効化

リエントラント

なし。

処理例の概要

モード設定：反転機能 ON

0 データのカラーは red (4)で指定されます。

1 データのカラーは blue (1)で指定されます。

カラー化処理については、入力画像が 1 つ、出力画像が 3 つ。

入力/出力画像とメモリの設定：

入力画像は IMAGE_BUF_SRC のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：256

出力画像は IMAGE_BUF_DEST[j]のバッファにあります。画像サイズ：H176xV176、メモリ水平サイズ：256

処理ブロック設定：

入力画像では、ブロックのブロック開始座標は(24,40)で、処理ブロックサイズ（水平サイズ 80、垂直サイズ 32）によって、カラー化処理が行われます。

開始ピクセルが(40, 40)の出力画像（R、G、B）に保存されます。

コード例

```
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_tR_GDT_Colorsyn (
       img_src_example,
       img_dest_example,
       blk_conv_info_example,
       gdt_colorsyn_para_cfg_example,
       p_callback*
)
```

特記事項

なし。

4. デモプロジェクト

本ドライバを使用したサンプルコード（デモプロジェクト）を用意しています。

以下のドキュメント番号にて弊社 WEB にて検索してください。

ドキュメント番号

1) r01an4755
2) r01an4810

5. 使用上の注意

5.1 画像データサイズ

画像データの水平サイズおよび垂直サイズは、8 の倍数(ビット)としてください。また、画像の水平サイズは、水平メモリサイズ以下に設定してください。画像フォーマット例を図 5-1 に示します。

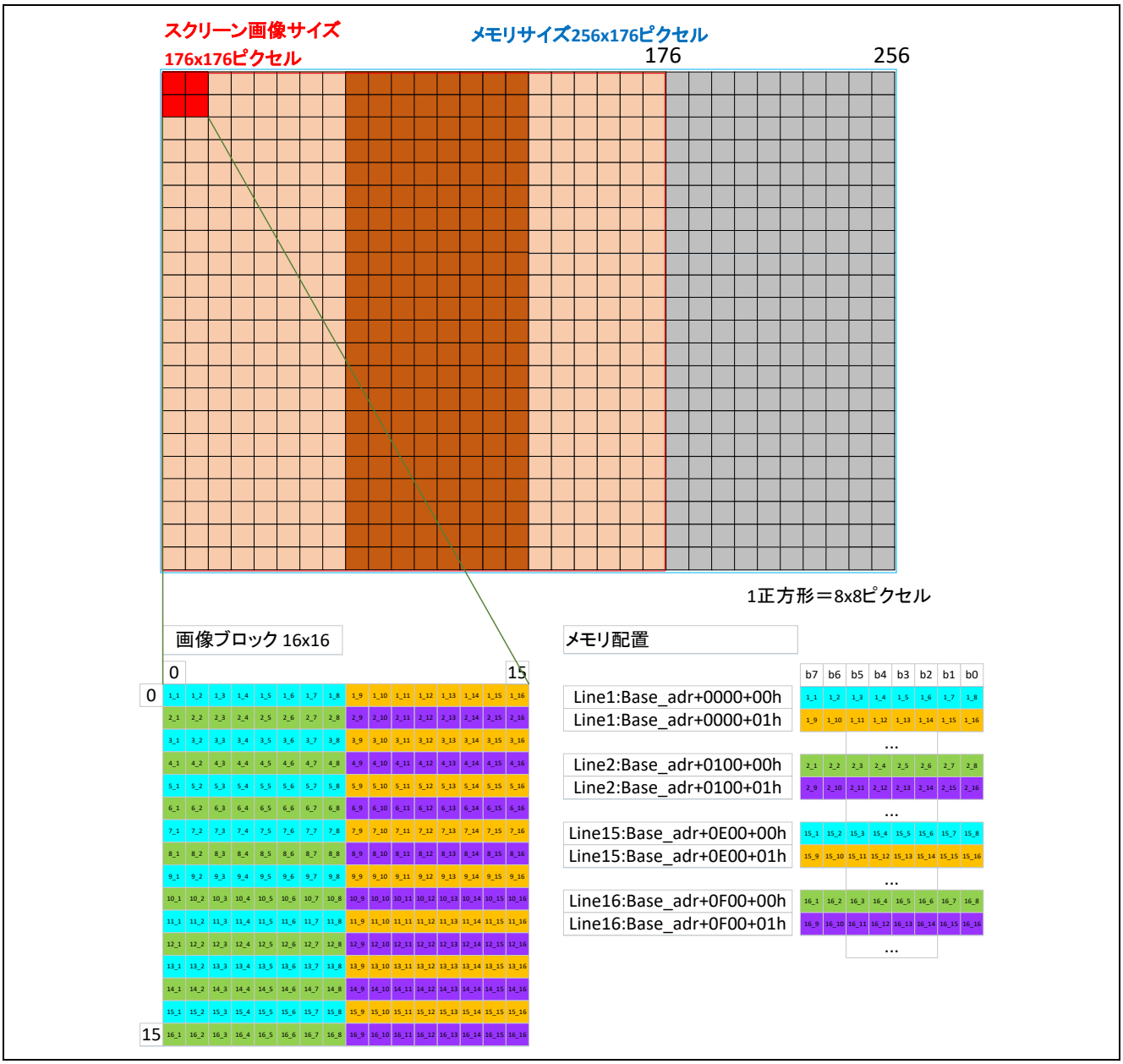

図 5-1　画像のフォーマット例

5.2　水平メモリサイズ

本ドライバはワーク領域のサイズを引数で指定する必要があります。加工する画像サイズ同等もしくは以上のサイズを次の中から選択してください。

水平メモリサイズは、32、64、128、または 256 を設定してください。ただし、カラーデータ整列機能の 3 ビットモード使用時は、96、192、384、または 768、4 ビットモード使用時は 128、256、512、または 1024 を設定してください。

5.3　画像処理単位

回転機能(R_GDT_Rotate)、モノクロ合成機能(R_GDT_Monochrome)、およびカラー合成機能(R_GDT_Colorsyn)は、画像処理単位を 2 種類(8×8 ビットまたは 16×16 ビット)から選択できます。各関数の画像処理データサイズ選択変数(gdtdsz)で設定してください。

回転機能を画像処理単位 16×16 ビットで使用する際、処理前および処理後ブロックの開始アドレスは偶数番地に設定してください。

モノクロ合成機能およびカラー合成機能では、画像処理単位 8×8 ビットのみ対応しています。これらの関数使用時は gdtdsz=1 に設定してください。

5.4　DMAC の割り込み設定および優先順位の制限

本 GDT ドライバは、必ず計 2 チャネルの DMAC を使用します。他の機能で使用する DMAC のチャネルと重複しないように、使用チャネルを設定してください。

また、GDT ドライバで使用する DMAC の割り込み優先順位に制限があります。GDT ドライバで使用する DMAC のチャネルは、R_GDT_DmacChSel 関数で設定します。この関数では、”GDT への入力データ転送用 DMAC”と、”出力データ転送用 DMAC”を設定します。上記で選択した DMAC のチャネルは、以下制限を満たすよう DMAC の割り込み優先順位を設定してください。

*制限：使用する DMAC 割り込みの優先順位

入力データ転送に使用するチャネル　＞　出力データ転送に使用するチャネル

また、DMAC の各チャネルの割り込み優先順位は、

/Device/Config/r_dmac_cfg.h のマクロ定義”DMACn_INT_PRIORITY(n=0~3)”

で設定できます。

例：入力にチャネル 1、出力にチャネル 2 を使う場合

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

6.　付録

6.1　　本ドライバの使用方法の詳細説明

この付録では、本ドライバの使用方法について詳述します。なお、1.2 章で本ドライバの使用方法を簡単に説明しているので参照してください。ここでは例として GDT の Rotate API を使用しますが、他の API に適用することも可能です。

ステップ **0**：データバッファを用意します

GDT ドライバを使用するには、少なくとも次の 3 つの変数が必要です。

- **画像データ**
- **作業領域用バッファ**
- **出力画像バッファ**

GDT ドライバには、水平サイズに関する制限があり、32、64、128、または 256 ピクセルを必要とします。

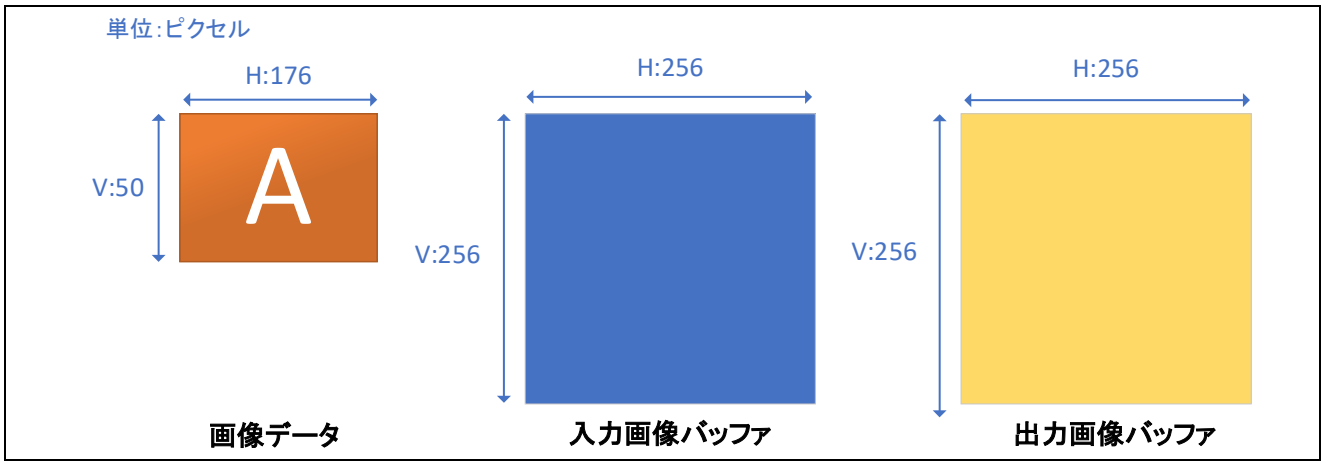

図 6-1　GDT ドライバに必要なデータ領域

ステップ **1**：入力画像データを作業領域用バッファにコピーします

ドライバの制限を回避するには、コピーをすることが必要です。元の画像データの水平サイズがドライバの許容サイズに合う場合は、このステップは不要です。

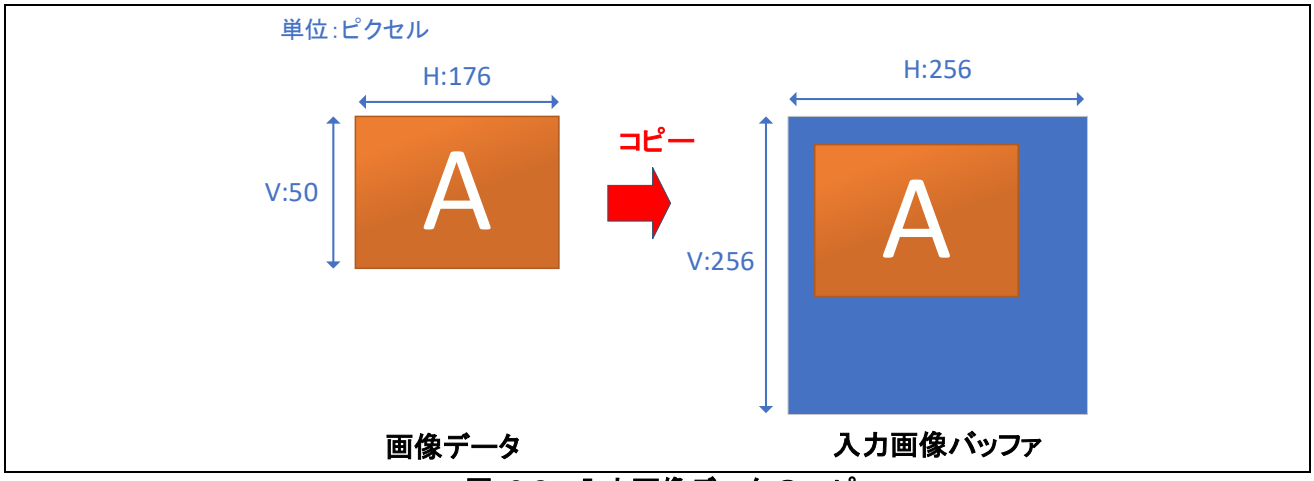

図 6-2　入力画像データのコピー

ステップ **2**：**GDT** 用の **API** 関数を呼び出します

ここでは、GDT API の Rotate 機能の引数について説明します。ただし、説明は入力と出力に対応する 1 番目と 2 番目の引数についてのみ行います。その他の引数については、3 章に示すコード例を参照してください。

図 6-3 を参照して API の引数を設定します。その後、API の呼び出しを実行します。

大別すると、GDT API には 3 つの情報が必要です。
「表示サイズ」、「バッファ情報」、「出力画像バッファ」です。

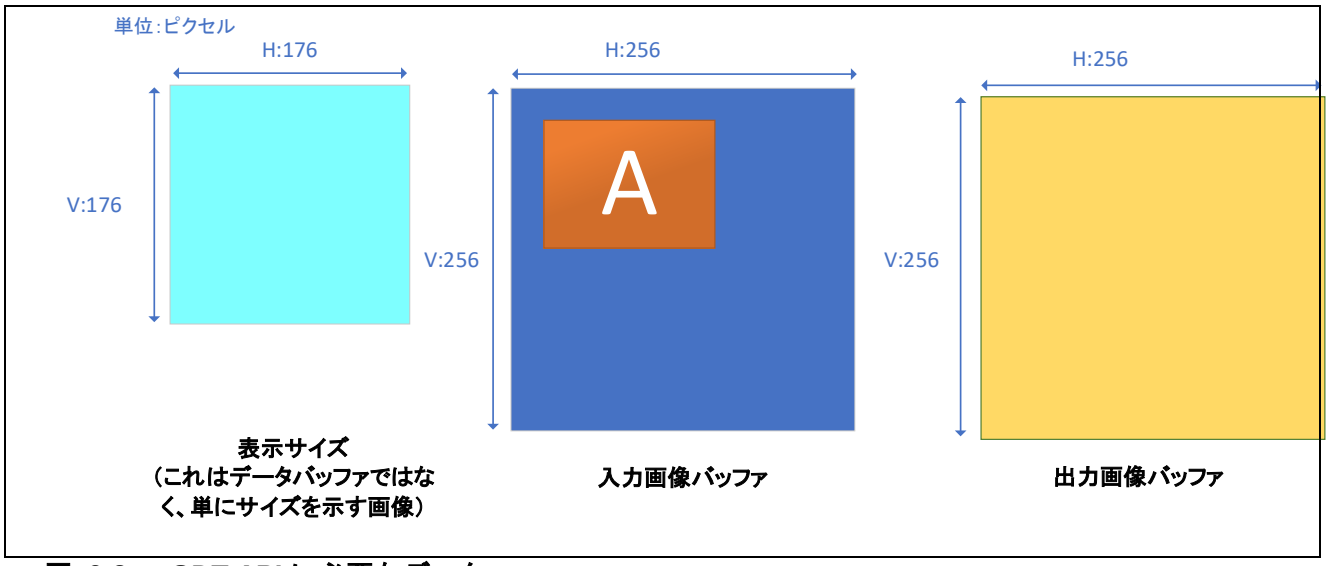

図 6-3　GDT API に必要なデータ

次に示すのは、3 章の API 関数からの引用です。

***R_GDT_Rotate API インターフェイス**

```
e_gdt_err_t  R_GDT_Rotate (
             st_img_in_info_t*          img_src,             /* 入力画像情報 */
             st_img_out_info_t*         img_dest,            /* 出力画像情報 */
             st_blk_conv_info_t*        blk_conv_info,       /* 変換ブロック情報 */
             st_gdt_rotate_para_cfg_t*  gdt_rotate_para_cfg, /* 回転機能の設定 */
             gdt_cb_event_t const       p_callback           /* コールバック関数*/
)
```

次の構造は、Rotate API の引数です。これらの色は、図 6-4 の色に対応しています。

***st_img_in_info、st_img_out_info、st_blk_conv_info_t の詳細構造**

```
typedef struct
{
    uint8_t  img_num;                          /*!< 入力画像の数 */
    uint32_t start_addr[IMG_IN_FRAME_NUM];     /*!< 入力画像のアドレス*/
    uint32_t size_h;                           /*!< 入力画像の水平サイズ*/
    uint32_t size_v;                           /*!< 入力画像の垂直サイズ*/
    uint32_t mem_size_h;                       /*!< 入力画像メモリの水平サイズ */
} st_img_in_info_t;

typedef struct
{
    uint8_t img_num;                           /*!< 出力画像の数       */
    uint32_t start_addr[IMG_OUT_FRAME_NUM];    /*!< 出力画像のアドレス    */
    uint32_t size_h;                           /*!< 出力画像の水平サイズ   */
    uint32_t size_v;                           /*!< 出力画像の垂直サイズ */
    uint32_t mem_size_h;                       /*!< 出力画像メモリの水平サイズ */
} st_img_out_info_t;

typedef struct
{
    uint32_t start_pix_h_src[IMG_IN_FRAME_NUM];    /*!< 入力画像の開始ピクセル、水平方向 */
    uint32_t start_pix_v_src[IMG_IN_FRAME_NUM];    /*!< 入力画像の開始ピクセル、垂直方向 */
```

```
    uint32_t start_pix_h_dest[IMG_OUT_FRAME_NUM]; /*!< 出力画像の開始ピクセル、水平方向 */
    uint32_t start_pix_v_dest[IMG_OUT_FRAME_NUM]; /*!< 出力画像の開始ピクセル、垂直方向 */
    uint32_t src_size_h;                          /*!< 変更領域のサイズ、水平方向      */
    uint32_t src_size_v;                          /*!< 変更領域のサイズ、垂直方向      */
} st_blk_conv_info_t;
```

図 6-4 は、図 6-3 に加える情報を示しています。

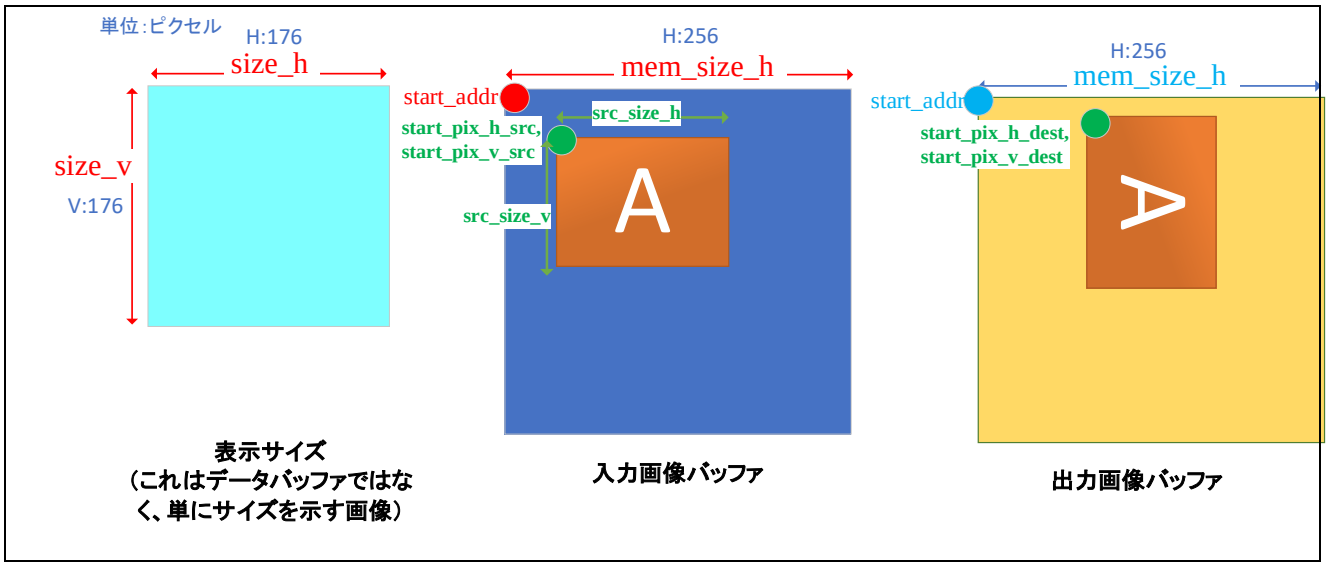

図 6-4　API の対応する引数

7.　参考ドキュメント

ユーザーズマニュアル：ハードウェア

RE01 1500KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0796
RE01 256KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0894

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RE01 グループ CMSIS Package スタートアップガイド

RE01 1500KB、256KB グループ CMSIS パッケージを用いた
開発スタートアップガイド R01AN4660

（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境

（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Oct.15.2019	-	初版発行
1.01	Nov.26.2019	プログラム プログラム	r_gdt_cfg.h の RAM/ROM 配置不備を修正 以下 2 つの内部関数が RAM に設定しても RAM に配置されない問題を改修 v_gdt_dmac_blk_upinf_in_array 関数 e_gdt_judge_cial_dest_mem_size 関数 r_gdt_api.h、r_gdt_cfg.h から未使用定義を削除
1.02	Dec.16.2019	- プログラム (256KB)	256KB グループに対応 256KB の IO デファインにあわせて修正
1.03	Feb.21.2020	プログラム (256KB, 1500KB)	内部関数 v_gdt_cpuline_byte_rd_gdt_limitの不備を修正

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。