TDE

# API Reference

**Issue** **05**

**Date** **2013-06-21**

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

# About This Document

## Purpose

This document describes the application programming interfaces (APIs), data types, and instances of the two-dimensional engine (TDE).

## Related Versions

The following table lists the product versions related to this document.

| Product Name | Version |
|---|---|
| Hi3531 | V100 |
| Hi3532 | V100 |
| Hi3521 | V100 |
| Hi3520A | V100 |
| Hi3518 | V100 |
| Hi3520D/Hi3515A | V100 |
| Hi3515A | V100 |
| Hi3515C | V100 |

## Intended Audience

This document is intended for:

- Technical support personnel
- Board development engineers

# Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

## Issue 05 (2013-06-21)

This issue is the fifth official release, which incorporates the following changes:

The descriptions of the Hi3515A are added.

## Issue 04 (2013-05-09)

This issue is the fourth official release, which incorporates the following changes:

**Chapter 1 Overview**

In section 1.2.2, the descriptions and precautions related to the **g_u32TdeTmpBufW** and **g_u32TdeTmpBufH** parameters are updated.

## Issue 03 (2013-04-03)

This issue is the third official release, which incorporates the following changes:

The descriptions of the Hi3515A are added.

## Issue 02 (2013-02-28)

This issue is the second official release, which incorporates the following changes:

The descriptions of the Hi3520D are added.

## Issue 01 (2012-08-30)

This issue is the first official release, which incorporates the following changes:

**Chapter 2 API Reference**

In section 2.1, the description of the **pDeflickerLevel** parameter is updated in the **Parameter** field of HI_TDE2_GetDeflickerLevel, the description of the **enDeflickerLevel** parameter is updated in the **Parameter** field of HI_TDE2_SetDeflickerLevel, and the descriptions are updated in the **Note** field of HI_TDE2_EnableRegionDeflicker.

**Chapter 3 Data Types**

In section 3.2, the descriptions are updated in the **Note** fields of TDE2_COLOR_FMT_E and TDE2_COLORKEY_U.

## Issue 00B60 (2012-08-09)

This issue is the eighth draft release, which incorporates the following changes:

**Chapter 1 Overview**

In section 1.2, the descriptions of the **g_u32TdeTmpBufW** and **g_u32TdeTmpBufH** parameters are updated.

**Chapter 2 API Reference**

In section 2.2, the descriptions are updated and Figure 2-7 is added in the **Note** field of HI_TDE2_Bitblit, the descriptions are updated in the **Note** field of HI_TDE2_Mb2Mb, and the APIs HI_TDE2_MbQuickCopy, HI_TDE2_MbRotate, and HI_TDE2_MbLDC are added.

**Chapter 3 Data Types**

In section 3.2, TDE2_ROTATE_ANGLE_E and LDC_ATTR_S are added.

## Issue 00B50 (2012-06-08)

This issue is the seventh draft release, which incorporates the following changes:

The descriptions of the Hi3520A are added.

## Issue 00B40 (2012-04-20)

This issue is the sixth draft release, which incorporates the following changes:

The descriptions of the Hi3521 are added.

## Issue 00B30 (2012-03-11)

This issue is the fifth draft release, which incorporates the following changes:

**Chapter 1 Overview**

Section 1.2 is added.

## Issue 00B20 (2012-02-15)

This issue is the fourth draft release, which incorporates the following changes:

**Chapter 2 API Reference**

In section 2.2, the API HI_TDE2_BeginJob or HI_TDE2_BeginVideoJob is added in the **Note** field of HI_TDE2_MbBlit, HI_TDE2_Mb2Mb, and HI_TDE2_MbFill.

## Issue 00B10 (2012-01-15)

This issue is the third draft release, which incorporates the following changes:

**Chapter 2 API Reference**

In section 2.2, the APIs HI_TDE2_BeginVideoJob, HI_TDE2_Mb2Mb, and HI_TDE2_MbFill are added.

**Chapter 3 Data Types**

In section 3.2, the data types TDE2_MBFILL_E and TDE2_CSC_OPT_S are added.

## Issue 00B02 (2011-12-07)

This issue is the second draft release, which incorporates the following changes:

**Chapter 2 API Reference**

In section 2.2, the **Note** field of HI_TDE2_Bitblit is updated, and the minimum minification is changed from 15 to 255.

**Chapter 3 Data Types**

In section 3.2, the **Syntax** and **Member** fields of TDE2_COLORKEY_COMP_S are updated, and the **Member** fields of TDE2_BLEND_MODE_E and TDE2_BLENDCMD_E are updated.

**Chapter 4 Error Codes**

In Table 4-1, the minimum minification is changed from 15 to 255.

## Issue 00B01 (2011-10-10)

This issue is the first draft release.

# Contents

# Figures

# Tables

# 1 Overview

## 1.1 Description

The two-dimensional engine (TDE) provides rapid graphics drawing functions through hardware when the on-screen display (OSD) function and graphical user interface (GUI) are used. Such functions contain rapid bitmap transfer, rapid color filling, rapid anti-flicker transfer, rapid bitmap scaling, point drawing, horizontal/vertical line drawing, bitmap format conversion, bitmap alpha blending, bitmap Boolean operation by bits, and colorkey.

## 1.2 Loading Drivers

### 1.2.1 Commands

To load a driver, run the following command:

```
insmod xx_tde.ko g_pszTdeMmzName="mmzname" g_u32TdeTmpBufW=800
g_u32TdeTmpBufH=600
```

where **xx** is the chip model such as Hi3531, Hi3532, Hi3521, Hi3520A, Hi3518, Hi3520D, Hi3515A, or Hi3515C. The parameter values can be changed.

### 1.2.2 Parameters

#### g_pszTdeMmzName

This parameter determines the media memory zone (MMZ) from which the internal memory used by the TDE is allocated. This parameter is a string. If a driver is loaded, the MMZ is defined. If this parameter is not set, the memory used by the TDE is allocated from an anonymous MMZ by default.

#### g_u32TdeTmpBufW and g_u32TdeTmpBufH

The **g_u32TdeTmpBufW** and **g_u32TdeTmpBufH** parameters define the size of the temporary buffer that is used when HI_TDE2_Osd2Mb or HI_TDE2_MbBlit is called. The default values of the two parameters are (800, 600). You can adjust the two parameters to specify the maximum size of the operation area in the source picture based on services. If you do not use HI_TDE2_Osd2Mb and HI_TDE2_MbBlit, set the two parameters to **0**.

**CAUTION**

- Because the Hi3518 does not support HI_TDE2_Osd2Mb and HI_TDE2_MbBlit, the default values of the two parameters are (0, 0), and setting the two parameters has no effect.
- For the Hi3520D ,Hi3515A, or Hi3515C the two parameters need to be set when the following two functions are used: transfer function (when the source operation area and destination operation area overlap and horizontal or vertical inverse scanning is required) and OverlayEx function.

# 1.3 Reference Field Description

## 1.3.1 API Reference Fields

This document describes the application programming interfaces (APIs) by using nine reference fields, as shown in Table 1-1.

**Table 1-1** Description of API reference fields

| Reference Field | Description |
|---|---|
| Purpose | Describes the major function of an API. |
| Syntax | Lists the required header files and the API prototype declaration when an API is called. |
| Parameter | Describes the parameters and attributes of an API. |
| Description | Describes the working process of an API. |
| Return Value | Lists the possible return values and their definitions of an API. |
| Requirement | Lists the header files of an API and the library files to be linked when the API is complied. |
| Note | Describes the precautions when an API is called. |
| Example | Lists the example of calling an API. |
| See Also | Lists the related APIs. |

## 1.3.2 Data Type Reference Fields

This document describes the data types by using five reference fields, as shown in Table 1-2.

**Table 1-2** Descriptions of data type reference fields

| Reference Field | Description |
|---|---|
| Description | Describes the major function of a data type. |
| Syntax | Lists the definition statement of a data type. |
| Member | Lists the members of a data type and the definition of each member. |
| Note | Lists the precautions when a data type is used. |
| See Also | Lists the related data types and interfaces. |

# 2 API Reference

## 2.1 API Description

The API reference of the TDE describes the operations related to 2D acceleration.

This module provides the following APIs:

- HI_TDE2_Open: Starts the TDE device.
- HI_TDE2_Close: Closes the TDE device.
- HI_TDE2_BeginJob: Creates a TDE task.
- HI_TDE2_BeginVideoJob: Creates a TDE video task.
- HI_TDE2_EndJob: Submits the TDE task to which operations are added.
- HI_TDE2_QuickCopy: Adds a rapid copy operation to a TDE task.
- HI_TDE2_QuickFill: Adds a rapid filling operation to a TDE task.
- HI_TDE2_QuickResize: Adds a raster bitmap scaling operation to a TDE task.
- HI_TDE2_QuickDeflicker: Adds a raster bitmap anti-flicker operation to a TDE task.
- HI_TDE2_GetDeflickerLevel: Obtains the anti-flicker level.
- HI_TDE2_SetDeflickerLevel: Sets the anti-flicker level.
- HI_TDE2_GetAlphaThresholdValue: Obtains the alpha judgment threshold.
- HI_TDE2_SetAlphaThresholdValue: Sets the alpha judgment threshold.
- HI_TDE2_GetAlphaThresholdState: Queries whether the alpha judgment function is enabled.
- HI_TDE2_SetAlphaThresholdState: Enables or disables alpha judgment.
- HI_TDE2_EnableRegionDeflicker: Enable or disables the regional anti-flicker function.
- HI_TDE2_Bitblit: Adds the transfer operation with additional functions performed on the raster bitmap to a TDE task.
- HI_TDE2_PatternFill: Fills a pattern.
- HI_TDE2_MbBlit: Adds the transfer operation with additional functions performed on the macroblock bitmap to a TDE task.
- HI_TDE2_SolidDraw: Adds the filling operation with additional functions performed on the raster bitmap to a TDE task.
- HI_TDE2_BitmapMaskRop: Adds the mask raster operation (ROP) operation performed on the raster bitmap to a TDE task.

- HI_TDE2_BitmapMaskBlend: Adds the mask blending operation performed on the raster bitmap to a TDE task.
- HI_TDE2_Mb2Mb: Adds the transfer operation with additional functions performed on the macroblock bitmap to a TDE video task
- HI_TDE2_MbFill: Adds a macroblock filling operation to a task.
- HI_TDE2_Osd2Mb: Adds the operation of blending raster data to the macroblock bitmap to a task.
- HI_TDE2_CancelJob: Cancels a specific TDE task.
- HI_TDE2_WaitForDone: Waits for the completion of a specific TDE task.
- HI_TDE2_MbQuickCopy: Adds an operation of rapidly copying macroblock bitmaps to a TDE task.
- HI_TDE2_MbRotate: Adds an operation of rotating macroblock bitmaps to a TDE task.
- HI_TDE2_MbLDC: Adds an operation of vertically correcting distorted macroblock bitmaps to a TDE task.

# 2.2 Function Reference

## HI_TDE2_Open

[Purpose]

To start the TDE device.

[Syntax]

```
HI_S32 HI_TDE2_Open(HI_VOID);
```

[Description]

This API is used to start the TDE device.

[Parameter]

None

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_SUCCESS | Success. |
| HI_ERR_TDE_DEV_OPEN_FAILED | The TDE device fails to be started. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Call this API to start the TDE device before performing operations on the TDE device.
- This API can be called repeatedly to start the TDE device.

[Example]

```
/*Declaration*/
HI_S32 s32Ret = 0;

/*Start the TDE device*/
s32Ret = HI_TDE2_Open();
if (HI_SUCCESS != s32Ret)
{
    return -1;
}

/*Close the TDE device*/
HI_TDE2_Close();
```

## HI_TDE2_Close

[Purpose]

To stop the TDE device.

[Syntax]

```
HI_VOID HI_TDE2_Close(HI_VOID);
```

[Description]

This API is used to stop the TDE device.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

The times of calling HI_TDE2_Open and HI_TDE2_Close must be the same.

[Example]

None

## HI_TDE2_BeginJob

[Purpose]

To create a TDE task.

[Syntax]

```
TDE_HANDLE HI_TDE2_BeginJob(HI_VOID);
```

[Description]

This API is used to create a TDE task. The TDE manages TDE commands as TDE tasks. A TDE task consists of a set of TDE commands. That is, a task may contain one or more TDE operations. Each TDE command corresponds to a TDE operation. After creating a TDE task and adding TDE operations, you can call HI_TDE2_EndJob to submit the TDE task. The TDE commands in a task are executed in sequence.

[Parameter]

None

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Ensure that the TDE device is started before calling this API.
- Ensure that a valid task handle is obtained by checking the return value.
- The TDE can buffer at most 200 tasks.

- HI_TDE2_EndJob must be called if HI_TDE2_BeginJob is called; otherwise, the memory is leaked.

[Example]

```
/* declaration */
    HI_S32  s32Ret;
    TDE_HANDLE s32Handle;

    /* create a TDE job */
    s32Handle = HI_TDE2_BeginJob();
    if(HI_ERR_TDE_INVALID_HANDLE == s32Handle
|| HI_ERR_TDE_DEV_NOT_OPEN == s32Handle)
    {
        return -1;
    }

    /* submit the job */
    s32Ret = HI_TDE2_EndJob(s32Handle, HI_FALSE, HI_TRUE, 20);
    if(HI_SUCCESS != s32Ret)
    {
        return -1;
}
```

## HI_TDE2_BeginVideoJob

[Purpose]

To create a TDE video task.

[Syntax]

```
TDE_HANDLE  HI_TDE2_BeginVideoJob(HI_VOID);
```

[Description]

This API is used to create a TDE video task. The TDE manages TDE commands as TDE tasks. A TDE task consists of a set of TDE commands and it may contain one or more TDE operations. Each TDE command corresponds to a TDE operation. After creating a TDE video task and adding TDE operations, you can call HI_TDE2_EndJob to submit the TDE video task. The TDE commands in a task are executed in sequence.

[Parameter]

None

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
| --- | --- |
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Ensure that the TDE device is started before calling this API.
- Ensure that a valid task handle is obtained by checking the return value.
- The TDE can buffer at most 200 tasks.
- HI_TDE2_EndJob must be called if HI_TDE2_BeginVideoJob is called; otherwise, the memory is leaked.
- You can add operations to a TDE video task by calling HI_TDE2_Mb2Mb, HI_TDE2_MbBlit, HI_TDE2_MbFill, HI_TDE2_Osd2Mb, HI_TDE2_MbRotate, HI_TDE2_MbLDC, or HI_TDE2_QuickCopy.

## HI_TDE2_EndJob

[Purpose]

To submit the created TDE task.

[Syntax]

```
HI_S32 HI_TDE2_EndJob(TDE_HANDLE s32Handle, HI_BOOL bSync, HI_BOOL bBlock,
                      HI_U32 u32TimeOut);
```

[Description]

This API is used to submit a TDE task. You can specify whether the API is called in block mode or non-block mode. If it is in block mode, you can set the timeout period.

- Block

  When the API is called, the API is not returned at once until one of the following conditions is met:

  - All commands of the TDE task are executed.
  - The waiting times out.
  - The waiting is terminated.
- Non-block

  After the API is called, the API is returned at once no matter whether the commands of the TDE task are executed.

You can set a maximum waiting period in block mode. If the waiting times out but the commands of the TDE task are not executed, the API is returned. The commands, however, are executed later.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| bSync | Whether to submit a TDE task in synchronous mode.<br>HI_TRUE: synchronous mode<br>HI_FALSE: non-synchronous mode | Input |
| bBlock | Block flag.<br>HI_TRUE: block<br>HI_FALSE: non-block | Input |
| u32TimeOut | Timeout period in the unit of jiffies (10 ms). | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_SUCCESS | The task is submitted successfully.<br>• Block task: All TDE commands of the task are executed.<br>• Non-block task: All TDE commands of the task are submitted successfully. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_JOB_TIMEOUT | The waiting times out. |
| HI_ERR_TDE_EMPTY_JOB | The task is empty. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.

- If you use the block mode, when HI_TDE2_EndJob is returned due to timeout or interruption, note that the operation continues till it is complete even though the API related to the TDE operation is returned in advance.
- After a task is submitted, its handle becomes invalid, and the error code HI_ERR_TDE_INVALID_HANDLE is returned if you submit this task again.
- You cannot submit a task without operations; otherwise, the error code HI_ERR_TDE_EMPTY_JOB is returned.

[Example]

None

## HI_TDE2_QuickCopy

[Purpose]

To add a rapid copy operation to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_QuickCopy(TDE_HANDLE s32Handle,
                         TDE2_SURFACE_S *pstSrc,
                         TDE2_RECT_S *pstSrcRect,
                         TDE2_SURFACE_S *pstDst,
                         TDE2_RECT_S *pstDstRect);
```

[Description]

This API is used to copy the specified area pstSrcRect in the bitmap pstSrc to the memory pstDst with the output area pstDstRect.

The bitmap, operation area, and the relationships between them are described as follows:

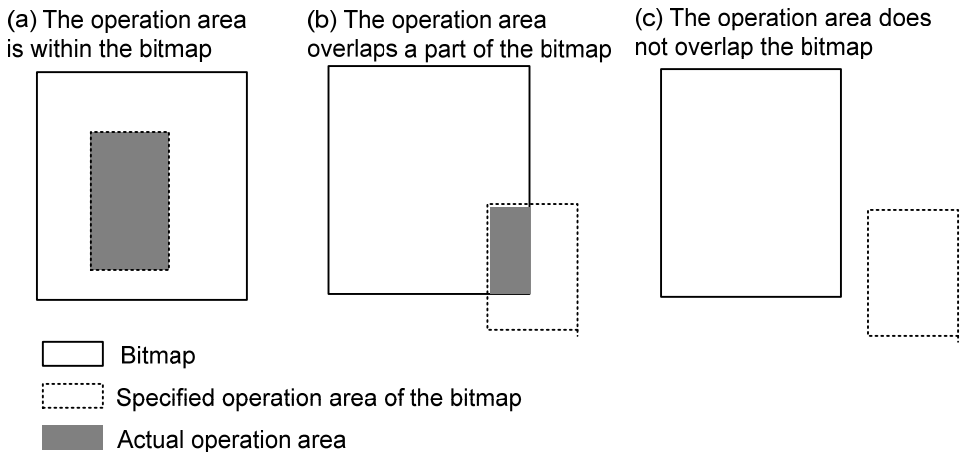
- The basic bitmap information is described by TDE2_SURFACE_S, including the pixel width, pixel height, stride between lines, color format, and physical address of the bitmap.
- The rectangle range of the bitmap relating to an operation, that is, operation area, is described by TDE2_RECT_S. The information contains the start position and rectangle size.
- Figure 2-1 shows the relationships between bitmaps and operation areas.

  By specifying the operation area, you can specify a part of the bitmap or the entire bitmap for an operation.
  - If you want to specify the entire bitmap, the start point of the operation area must be (0, 0) and the width and height must be the same as those of the bitmap.
  - If you want to specify a part of the bitmap, specify the size of the operation area. As shown in part (a) of Figure 2-1, the specified area is the valid operation area. Note: If the specified operation area overlaps a part of the bitmap (as shown in part (b) of Figure 2-1), the specified operation area is clipped automatically. Therefore, the valid operation area is the gray overlapped part.
  - If the specified operation area does not overlap the bitmap (as shown in part (c) of Figure 2-1), it indicates that the configuration is incorrect and the error code HI_ERR_TDE_INVALID_PARA is returned.

**NOTE**

The valid operation area refers to the overlapped part of the specified operation area and the bitmap.

**Figure 2-1** Relationships between bitmaps and operation areas




[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |

| Error Code | Description |
| --- | --- |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- The function of HI_TDE2_QuickCopy is implemented by using DMA transfer; therefore, HI_TDE2_QuickCopy is superior to HI_TDE2_Bitblit in the transfer function.
- The rapid copy operation does not support format conversion; therefore, ensure that the format of the source bitmap is the same as that of the target bitmap.
- The rapid copy operation does not support the scaling function. If the operation area size of the source bitmap is different from that of the target bitmap, the minimum common operation area in the two bitmaps is copied and transferred.
- Ensure that the specified operation area and the specified bitmap have a common area; otherwise, an error is returned. This requirement is applicable to other APIs.
- If the pixel format of a bitmap is greater than or equal to a byte, the base address and stride of the bitmap format must be aligned based on the pixel format. If the pixel format of a bitmap is smaller than a byte, the base address and stride of the bitmap must be aligned based on byte. This requirement is applicable to other APIs.
- If the pixel format of a bitmap is smaller than a byte, the horizontal start point and width of the bitmap must be aligned based on pixel.
- The horizontal start point and width of the YCbCr422 bitmap must be even numbers. This requirement is applicable to other APIs.

[Example]

None

## HI_TDE2_QuickFill

[Purpose]

To add a rapid filling operation to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_QuickFill(TDE_HANDLE s32Handle, TDE2_SURFACE_S *pstDst,
                         TDE2_RECT_S *pstDstRect, HI_U32 u32FillData);
```

[Description]

This API is used to fill u32FillData to the memory with the destination address pstDst and the output area pstDstRect, achieving the color filling function.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| u32FillData | Fill data. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- After this API is called, u32FillData is filled in the specified area in the bitmap directly. If you want to fill blue in a specified bitmap, specify a fill value corresponding to the blue color according to the bitmap format.
- If the bitmap format is ARGB1555 and the fill color is blue, set u32FillData to 0x801F (the alpha bit is 1).
- Hi3518 does not support this API.

[Example]

None

## HI_TDE2_QuickResize

[Purpose]

To add a raster bitmap scaling operation to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_QuickResize(TDE_HANDLE s32Handle,
                           TDE2_SURFACE_S *pstSrc,
                           TDE2_RECT_S *pstSrcRect,
                           TDE2_SURFACE_S *pstDst,
                           TDE2_RECT_S *pstDstRect);
```

[Description]

This API is used to scale down the specified area pstSrcRect in the bitmap pstSrc to the size of pstDstRect and copy the result to the memory pstDst with the output area pstDstRect at the same time.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null.. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |

| Error Code | Description |
|---|---|
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: hitde.a

[Note]

- The minification is less than 255, and there is no limitation on up scaling.
- You can scale the bitmap that serves as both source bitmap and target bitmap. If the memory of the source bitmap overlaps that of the target bitmap, the bitmaps cannot be scaled.
- If the formats of the source bitmap and target bitmap are different, a format is converted automatically.
- The Hi3518 does not support this API.

[Example]

None

## HI_TDE2_QuickDeflicker

[Purpose]

To add an anti-flicker operation to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_QuickDeflicker(TDE_HANDLE s32Handle,
                              TDE2_SURFACE_S *pstSrc,
                              TDE2_RECT_S *pstSrcRect,
                              TDE2_SURFACE_S *pstDst,
                              TDE2_RECT_S *pstDstRect);
```

[Description]

This API is used to perform the anti-flicker operation on the specified area pstSrcRect in the bitmap pstSrc and copy the result to the memory pstDst with the output area pstDstRect at the same time.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- The anti-flicker operation supports vertical filtering only.
- The anti-flicker operation can be performed on the source bitmap and target bitmap that are stored in the same memory.
- If the sizes of the specified input area and the output area are different, it is scaled down.
- If the formats of the source bitmap and target bitmap are different, a format is converted.

[Example]

None

## HI_TDE2_GetDeflickerLevel

[Purpose]

To obtain the anti-flicker level.

[Syntax]

```
HI_S32 HI_TDE2_GetDeflickerLevel(TDE_DEFLICKER_LEVEL_E *pDeflickerLevel);
```

[Description]

This API is used to obtain the anti-flicker level.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| pDeflickerLevel | Pointer to the enumeration of anti-flicker levels. | Output |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_SetDeflickerLevel

[Purpose]

To set the anti-flicker level.

[Syntax]

```
HI_S32 HI_TDE2_SetDeflickerLevel(TDE_DEFLICKER_LEVEL_E enDeflickerLevel);
```

[Description]

This API is used to set the anti-flicker level.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| enDeflickerLevel | Enumeration of anti-flicker levels. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_GetAlphaThresholdValue

[Purpose]

To obtain the alpha judgment threshold.

[Syntax]

```
HI_S32 HI_TDE2_GetAlphaThresholdValue(HI_U8 *pu8ThresholdValue);
```

[Description]

This API is used to obtain the alpha judgment threshold and is applicable when the result picture is in ARGB1555 format. If the alpha operation result of the foreground bitmap and background bitmap is less than the threshold, the alpha bit of the result pixel is 0; if the alpha operation result is greater than or less than the threshold, the alpha bit is 1.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| pu8ThresholdValue | Pointer to the alpha judgment threshold. | Output |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_SetAlphaThresholdValue

[Purpose]

To set the alpha judgment threshold.

[Syntax]

```
HI_S32 HI_TDE2_SetAlphaThresholdValue(HI_U8 u8ThresholdValue);
```

[Description]

This API is used to set the alpha judgment threshold. When a bitblit operation is performed on the foreground and background bitmaps, an intermediate bitmap in ARGB888 format is generated regardless of the formats of the foreground and background bitmaps. If the target picture is in ARGB1555 format and the alpha operation result of the foreground bitmap and background bitmap is less than the threshold, the alpha bit of the result pixel is 0; if the target picture is in ARGB1555 format and the alpha operation result is greater than or less than the threshold, the alpha bit is 1.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| u8ThresholdValue | Alpha judgment threshold. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_GetAlphaThresholdState

[Purpose]

To query whether the alpha judgment function is enabled.

[Syntax]

```
HI_TDE2_GetAlphaThresholdState(HI_BOOL * p_bEnAlphaThreshold);
```

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| p_bEnAlphaThreshold | Pointer to the status of the alpha judgment function | Output |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_SetAlphaThresholdState

[Purpose]

To enable or disable alpha judgment. When alpha judgment is enabled, the alpha judgment threshold is the user-defined value; when alpha judgment is disabled, the threshold is 0xFF.

[Syntax]

```
HI_TDE2_SetAlphaThresholdState(HI_BOOL bEnAlphaThreshold);
```

[Description]

This API is used to enable or disable alpha judgment.

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| bEnAlphaThreshold | Status of the alpha judgment function.<br>• True: The alpha judgment function is enabled.<br>• False: The alpha judgment function is disabled. | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
| --- | --- |
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

None

[Example]

None

## HI_TDE2_EnableRegionDeflicker

[Purpose]

To enable or disable the regional anti-flicker function.

[Syntax]

```
HI_S32 HI_TDE2_EnableRegionDeflicker(HI_BOOL bRegionDeflicker);
```

[Description]

This API is used to enable or disable the regional anti-flicker function.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| bRegionDeflicker | Regional anti-flicker enable flag.<br>• True: enabled<br>• False: disabled | Input |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

If anti-flicker is performed on a specific region by calling HI_TDE2_QuickDeflicker or HI_TDE2_Bitblit when regional anti-flicker is disabled, the values of the pixels around the region are not referenced. If regional anti-flicker is enabled, the values of the pixels around the region are referenced. Therefore, the anti-flicker results for region edges when anti-flicker is enabled are different from those when the regional anti-flicker is disabled. If anti-flicker is performed on an entire picture, the results obtained when anti-flicker is enabled are the same as those obtained when anti-flicker is disabled.

[Example]

None

## HI_TDE2_Bitblit

[Purpose]

To add the transfer operation with additional functions performed on the raster bitmap to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_Bitblit(TDE_HANDLE s32Handle,
                       TDE2_SURFACE_S *pstBackGround,
                       TDE2_RECT_S *pstBackGroundRect,
                       TDE2_SURFACE_S *pstForeGround,
                       TDE2_RECT_S *pstForeGroundRect,
                       TDE2_SURFACE_S *pstDst,
                       TDE2_RECT_S *pstDstRect,
                       TDE2_OPT_S *pstOpt);
```

[Description]

This API is used to perform operations on the specified area (pstForeGroundRect) of the foreground bitmap (pstForeGround) and the specified area (pstBackGroundRect) of the background bitmap (pstBackGround), and then copy the obtained bitmap to the specified area (pstDstRect) of the target bitmap (pstDst). The size of the specified area (pstBackGroundRect) of the background bitmap (pstBackGround) must be the same as the size of the specified area (pstDstRect) of the target bitmap (pstDst).

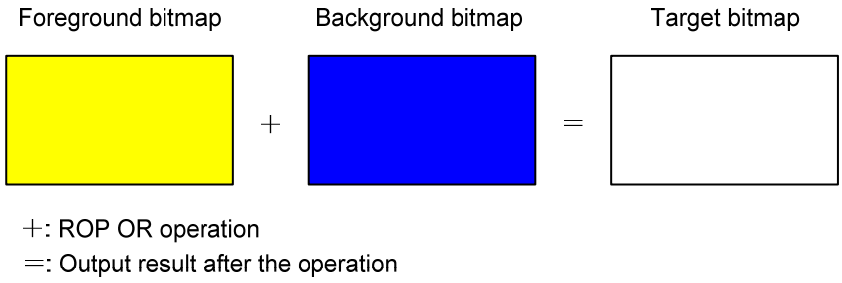
TDE2_OPT_S stores the configurations of the TDE operation function. For example, whether to perform the ROP operation and run the ROP command code; whether to specify the colorkey and set the value of the colorkey; whether to clip an area and specify the area to be clipped; whether to scale; whether to perform anti-flicker; whether to mirror; and whether to perform alpha blending. These operations can be simultaneously enabled.

The concepts related to the configuration items of TDE2_OPT_S are described as follows:

- Bitwise boolean operation, that is, ROP

  The ROP operation refers to the bitwise boolean operation (including bitwise AND and bitwise OR) that is performed on the RGB components and alpha components of the foreground bitmap and the background bitmap. After the operation, results are output. See Figure 2-2.

**Figure 2-2** Transfer operation during the ROP operation (src1: R, G, B = 0xFF, 0xFF, 0; src2: R, G, B = 0, 0, 0xFF)



- Alpha blending

  Alpha blending refers to the weight sum of the pixels of the foreground bitmap and the background bitmap based on the alpha value of the foreground bitmap. In this way, a bitmap with blended alpha value is obtained and the two bitmaps are blended with certain transparency. The alpha value of the output bitmap depends on the configured alpha blending command. For details, see the description of TDE2_BLENDCMD_E. There are two blending modes:

  **NOTE**

  The global alpha must be blended in either of modes.

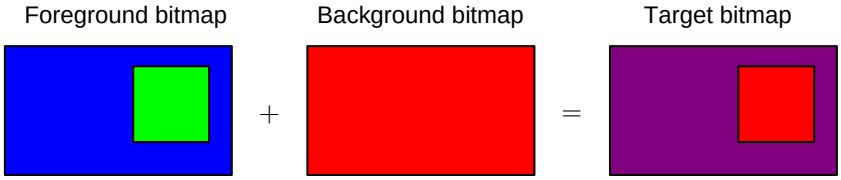
  - If the data of the foreground or background bitmap is premultiplied by alpha, select the foreground or background premultiplied alpha blending mode.
  - If the data of the foreground or background bitmap is not premultiplied, select the foreground or background non-premultiplied alpha blending mode.

- Colorkey operation

  The colorkey operation refers to that the pixels within the colorkey range are excluded from the TDE operations. You need to set the filtering conditions for each component based on the pixel format in colorkey settings. If all components of a color meet the filtering conditions, the color is a colorkey. There are two colorkey operation modes:

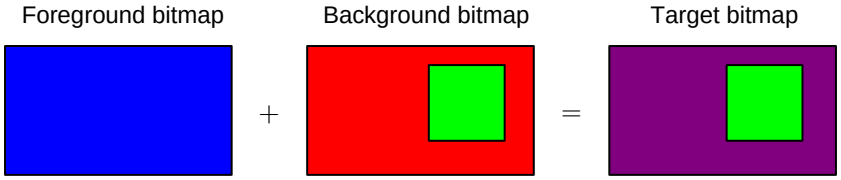
  - Performing the colorkey operation on the foreground bitmap: In this mode, the colorkey of the foreground bitmap is excluded from the colorkey operation and the background bitmap is retained. That is, the corresponding area in the background bitmap is copied to the output bitmap, as shown in Figure 2-3.
  - Performing the colorkey operation on the background bitmap: In this mode, the colorkey area in the background bitmap is copied to the output bitmap and the other areas are the operation results, as shown in Figure 2-4.

**Figure 2-3** Transfer operation during the colorkey operation performed on the foreground bitmap




+: Perform colorkey operation on the foreground bitmap and alpha operation
=: Output result after the operation

**Figure 2-4** Transfer operation during the colorkey operation performed on the background bitmap




+: Perform colorkey operation on the foreground bitmap and alpha operation
=: Output result after the operation

- Scaling operation

  When the sizes of the operation areas of the foreground bitmap and the target bitmap are different, perform one of the following two operations:

  - If the bResize parameter of TDE2_OPT_S is set to TRUE, scale the operation area in the foreground bitmap to the size of the operation area in the target bitmap and then perform other operations on the obtained foreground bitmap and the background bitmap.
  - If the bResize parameter of TDE2_OPT_S is set to FALSE, the operation area in the foreground bitmap is not scaled. Instead, the minimum area between the operation areas (pstForeGroundRect, pstBackGroundRect, and pstDstRect) of the foreground bitmap, background bitmap, and target bitmap is selected and served as the actual operation area in the three bitmaps.

- Anti-flicker operation

  The anti-flicker operation refers to that anti-flicker is performed on the operation area in the foreground bitmap and then other operations such as alpha blending operation are performed on the foreground bitmap and the background bitmap. You can determine whether to perform the anti-flicker operation by configuring the bDeflicker parameter of TDE2_OPT_S.

- Mirror function

  The mirror function refers to that the output result is reversed horizontally and/or vertically. You can specify the mirror type by configuring the enMirror parameter of TDE2_OPT_S. The mirror types are as follows:

  - Horizontal mirror: Symmetrically copy the output result in the horizontal direction.
  - Vertical mirror: Symmetrically copy the output result in the vertical direction.
  - Horizontal and vertical mirror: Symmetrically copy the output result in both horizontal and vertical directions.

- Color extension or correction function

  The color extension function refers to that the color with low precision is extended to the true color through the palette (also called CLUT). For example, if a CLUT8 bitmap has only 256 colors, you can construct a proper CLUT and then set the pu8ClutPhyAdd attribute of the bitmap surface to the start address of the CLUT. Then the TDE can implement the extension from CLUT8 to the true color ARGB by retrieving the CLUT.

  To implement color extension, you need to configure the following items:

  - CLUT start address pu8ClutPhyAddr of the bitmap surface. The memory in this address must be continuous.
  - bYCbCrClut item of the bitmap surface. This item specifies whether the CLUT is in the RGB space or YC space.
  - bClutReload item of TDE2_OPT_S. This item specifies whether the hardware needs to reload the CLUT. The CLUT reload flag needs to be marked when the color is extended from the CLUT to the RGB/AYCbCr for the first time.

- Clip function for the output picture

  - Generally, the pictures processed by the TDE are output to the specified area in the target bitmap. Through the clip function, only the specified part of the picture is output to the target bitmap. That is, the output picture is clipped and then output. The TDE supports the following two clip modes:
  - Intra-area clip: In this mode, the TDE operation result is the updated area within the clipped area. As shown in Figure 2-5, the clipped area overlaps the operation area in the target bitmap. Through the intra-area clip function, only the updated gray area is the TDE operation result and the other part of the destination operation area remains.

- Intra-area clip: The TDE operation result is the updated area outside of the clipped area. As shown in Figure 2-6, the clipped area overlaps the operation area in the target bitmap. Through the extra-area clip function, only the updated gray area is the TDE operation result and the part within the clipped area remains.

**Figure 2-5** Intra-area clip

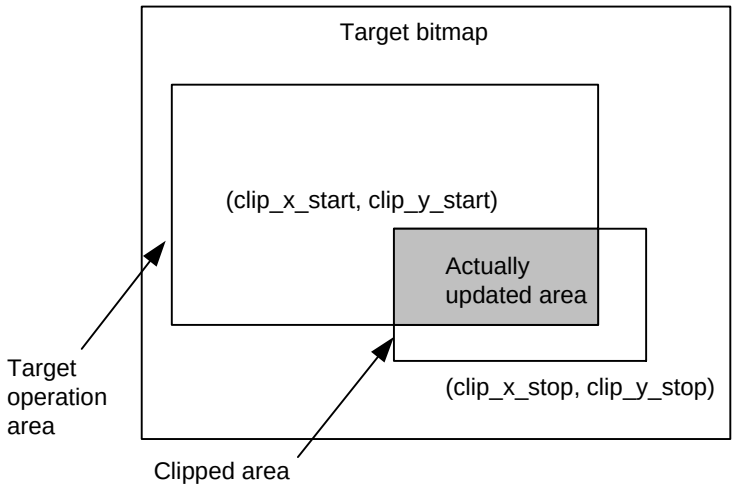
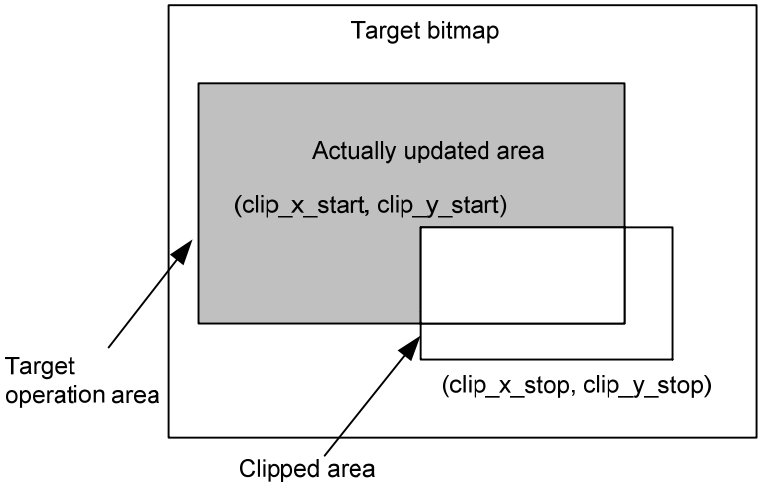



**Figure 2-6** Extra-area clip




- Output source of the alpha

  There are four output sources:

  - From the operation result
  - From the foreground bitmap
  - From the background bitmap
  - From the global alpha

**NOTE**

You need to select the source of operation result for the alpha blending operation.

- Single-source or dual-source graphics operation

  The single-source operation refers to that only one bitmap source is specified. For example, when only the background bitmap and target bitmap are specified, the foreground bitmap is null. In this case, you can perform the following operations on the background bitmap:

  - Bitmap transfer
  - Bitmap format conversion
  - Bitmap scaling
  - Bitmap anti-flicker
  - Bitmap color extension or correction
  - Bitmap output result clipping

  The dual-source operation refers to that two bitmap (background bitmap and foreground bitmap) sources are specified, and then the operation result of the two bitmaps are output to the specified area in the target bitmap. Here, the background bitmap can be the same as the target bitmap. The description of the operation is as follows: operate the foreground bitmap and the background bitmap and output the result to the background bitmap. The double-source operations are as follows:

  - ROP between the foreground bitmap and the background bitmap
  - Alpha blending operation between the foreground bitmap and the background bitmap
  - Colorkey operation
  - After performing the scaling or anti-flicker operation on the specified area in the foreground bitmap, perform the alpha blending operation between the obtained foreground bitmap and the background bitmap.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstBackGround | Background bitmap. | Input |
| pstBackGroundRect | Operation area in the background bitmap. | Input |
| pstForeGround | Foreground bitmap. | Input |
| pstForeGroundRect | Operation area in the foreground bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| pstOpt | Operation parameter settings. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

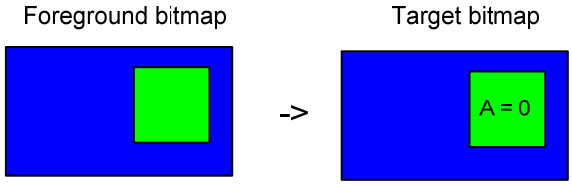
- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- The color space of the target bitmap must be the same as that of the background bitmap. The color space of the foreground bitmap can be different from that of the background or target bitmap; if so, the color space is converted.
- When the size of the foreground bitmap is different from that of the target bitmap, if you enable the scaling function, the bitmap is scaled based on the preset area; otherwise, the clip and transfer operations are performed based on the minimum value of the minimum common area.
- The global alpha, Alplh0, and Alpha1 range from 0 to 255.
- The background bitmap and the target bitmap can be the same.
- If you need only the single-source transfer operation (for example, performing the ROP and reverse operations on the source bitmap only), you can set null pointers for the foreground, background, pstForeGroundRect, and pstBackGroundRect. The foreground or background describes the bitmap and pstForeGroundRect or pstBackGroundRect describes the operation area.
- When the mirror function is enabled, the scaling function is disabled.

- For an inter-area clip operation, the clipped area must overlap the operation area; otherwise, an error code is returned. For an intra-area clip operation, the operation area cannot be completely overlaid with the clipped area; otherwise, an error code is returned. That is, the actually updated area cannot be blank.
- The CLUT reload flag needs to be marked when the color is extended from the CLUT to the RGB/AYCbCr for the first time.
- During the ROP operation, you can specify the color component and alpha component for the ROP operation by configuring the members enRopCode_Color and enRopCode_Alpha of TDE2_OPT_S respectively. For the ROP type, S1 indicates the background bitmap pstBackGround and S2 indicates the foreground bitmap pstForeGround.
- The Hi3518 supports only single-source operations.
- The Hi3518 does not support ROP, alpha blending, scaling, color extension, color correction, and clip by calling this API.
- The Hi3518 colorkey differs from the colorkey described in the **Description** field. The difference is that the alpha value of the Hi3518 colorkey is 0. Other component values of the Hi3518 colorkey are the same as those of the colorkey described in the **Description** field. See Figure 2-7.
- The Hi3520D/Hi3515A/Hi3515C does not support ROP and mirror.

**Figure 2-7** Transfer operation during the colorkey operation performed on the foreground bitmap for the Hi3518




->: indicates the colorkey operation.
A: indicates the alpha component.

[Example]

None

## HI_TDE2_PatternFill

[Purpose]

To fill a pattern.

[Syntax]

```
HI_S32 HI_TDE2_PatternFill(TDE_HANDLE s32Handle,
                           TDE2_SURFACE_S *pstBackGround,
                           TDE2_RECT_S *pstBackGroundRect,
                           TDE2_SURFACE_S *pstForeGround,
                           TDE2_RECT_S *pstForeGroundRect,
```

```
                                    TDE2_SURFACE_S *pstDst,
                                    TDE2_RECT_S *pstDstRect,
                                    TDE2_PATTERN_FILL_OPT_S *pstOpt);
```

[Description]

When the specified area pstForeGroundRect of the foreground bitmap pstForeGround is tiled onto the specified area pstBackGroundRect of the background bitmap pstBackGround, the operations including colorkey, ROP, clipping, color extension, and bitmap format conversion can be implemented. The operation result is transferred to the specified area pstDstRect of the target bitmap pstDst. When the background bitmap is filled with the foreground bitmap, the specified area in the foreground bitmap is scaled and the foreground bitmap is tiled onto the entire specified area in the background bitmap. If the specified area in the foreground bitmap is larger than the specified area in the background bitmap, the specified area in the foreground bitmap is automatically clipped.

- In single-source operation mode, the background bitmap and its specified area or the foreground bitmap and its specified area can be set to null. In this case, the foreground bitmap or background bitmap can be tiled onto the specified area in the target bitmap. During the tile process, you can convert the bitmap format, extend or correct the bitmap color, or clip the output bitmap.
- In dual-source operation mode, when the specified area in the background bitmap is filled with the specified area in the foreground bitmap, an operation is performed on the two bitmaps, and the operation result is output to the specified area in the target bitmap. The double-source operations are as follows:
  - ROP between the foreground bitmap and the background bitmap
  - Alpha blending between the foreground bitmap and the background bitmap
  - Colorkey operation

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstBackGround | Background bitmap. | Input |
| pstBackGroundRect | Operation area in the background bitmap. | Input |
| pstForeGround | Foreground bitmap. | Input |
| pstForeGroundRect | Operation area in the foreground bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| pstOpt | Operation parameter settings. | Input |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- When the background bitmap is null:
  - The specified area in the foreground bitmap is clipped if the specified area in the foreground bitmap is larger than the specified area in the target bitmap.
  - The width and height of the foreground bitmap must be even numbers unless the width and height are 1. There is no such requirement when the background bitmap is not null.
- If the following conditions are met, the size of the specified area in the background bitmap must be the same as that of the specified area in the target bitmap
  - The width and height of the specified area in the background bitmap are less than or equal to the maximum width and height of the background bitmap.
  - The width and height of the specified area in the target bitmap are less than or equal to the maximum width and height of the target bitmap

  If the width or height of the specified area in the target bitmap is greater than the maximum width or height of the target bitmap, the specified area is automatically clipped.

  If the width or height of the foreground bitmap is greater than the maximum width or height of the foreground bitmap or the width or height of the background bitmap is greater than the maximum width or height of the background bitmap, the foreground bitmap or background bitmap is not clipped and the format filling fails.

- If the specified area in the foreground bitmap is larger than the specified area in the target bitmap, the specified area in the foreground bitmap is automatically clipped.
- If the pixel format of the background bitmap is CLUT, the pixel format of the target bitmap must be CLUT. If the pixel format of the background bitmap is not CLUT, the pixel format of the target bitmap cannot be CLUT.
- That is, if the background bitmap is in other pixel formats, the target bitmap can be in the pixel format other than CLUT. In addition, the color spaces of the background bitmap and target bitmap can be different.
- The formats of the source bitmap and target bitmap cannot be byte.
- If both the foreground bitmap and background bitmap are not null, the operations including scaling, anti-flicker, and mirror are unavailable when the specified area in the background bitmap is filled with the specified area in the foreground bitmap. Other operations are the same as those performed on the two bitmaps during the

Bitblit process.

- When you clip an area, note that the clipped area must overlap the operation area; otherwise, an error occurs.
- The CLUT reload flag needs to be marked when the color is extended from CLUT to RGB/AYCbCr for the first time.
- During the ROP operation, you can specify the color component and alpha component for the ROP operation by configuring the members enRopCode_Color and enRopCode_Alpha of TDE2_OPT_S respectively. For the ROP type, S1 indicates the background bitmap pstBackGround and S2 indicates the foreground bitmap pstForeGround.
- Hi3518 does not support this API.
- The Hi3520D/Hi3515A/Hi3515C does not support the ROP.

[Example]

None

## HI_TDE2_MbBlit

[Purpose]

To add the transfer operation with additional functions performed on the macroblock bitmap to a TDE task. That is, the luminance macroblock data and the chrominance macroblock data are combined into raster data. During the combination, the scaling, anti-flicker, and clip operations can be performed concurrently.

[Syntax]

```
HI_S32 HI_TDE2_MbBlit(TDE_HANDLE s32Handle,
                      TDE2_MB_S *pstMB, TDE2_RECT_S *pstMbRect,
                      TDE2_SURFACE_S *pstDst, TDE2_RECT_S*pstDstRect,
                      TDE2_MBOPT_S *pstMbOpt);
```

[Description]

The luminance data and chrominance data of the specified area on the macroblock surface are combined into raster data and then output to the specified area on the destination surface. During the combination, the scaling function can be performed and the scaling mode is specified by the parameter enResize of pstMbOp. If scaling is not specified, the combined macroblock data is directly output to the destination surface and the excessive area is clipped. If the clip function is enabled, the clip and copy operations are performed. The anti-flicker function is also supported during the combination.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstMB | Surface of the macroblock. | Input |
| pstMbRect | Operation area in the macroblock. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |

| Parameter | Description | Input/Output |
|---|---|---|
| pstMbOpt | Attributes of the macroblock operation. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob or HI_TDE2_BeginVideoJob to obtain a valid task handle.
- For an YCbCr422 macroblock, if horizontal sampling is performed, the horizontal coordinate of the start point of the operation area must be an even number. This is no such restriction if vertical sampling is performed.
- The target bitmap can be YCbCr or RGB color space.
- Hi3518 does not support this API.

[Example]

None

## HI_TDE2_SolidDraw

[Purpose]

To add the filling operation with additional functions performed on the raster bitmap to a TDE task. The functions of drawing a point, drawing a line, filling a color block, and filling a memory on the surface can be implemented.

[Syntax]

```
HI_S32 HI_TDE2_SolidDraw(TDE_HANDLE s32Handle,
                         TDE2_SURFACE_S *pstForeGround,
                         TDE2_RECT_S *pstForeGroundRect,
                         TDE2_SURFACE_S *pstDst,
                         TDE2_RECT_S *pstDstRect,
                         TDE2_FILLCOLOR_S *pstFillColor,
                         TDE2_OPT_S *pstOpt);
```

[Description]

This API is used to operate the operation area of the foreground surface and the fill color and then output the result to the operation area of the destination surface. The operation can be alpha blending or ROP operation, during which the clip operation is supported.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstForeGround | Foreground bitmap. | Input |
| pstForeGroundRect | Operation area in the foreground bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| pstFillColor | Fill color. | Input |
| pstOpt | Operation attributes. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- When the foreground bitmap and the parameter pstOpt are null, the single color fill function can be implemented by calling HI_TDE2_SolidDraw. In this case, the functions of HI_TDE2_SolidDraw and HI_TDE2_QuickFill are the same. HI_TDE2_SolidDraw is called as follows:

  ```
  HI_TDE2_SolidDraw(s32Handle,NULL,NULL,pstDst,pstDstRect,
  pstFillColor,NULL);
  ```

- When the foreground bitmap is null, but the alpha blending or ROP operation is specified by the parameter pstOpt, HI_TDE2_SolidDraw can be used to perform alpha blending or ROP operation on the specified area in the target bitmap pstDst and fill color and then output the result to the target bitmap. That is, when the single color fill operation is performed, the transparent blending or ROP operation can be performed on the fill color and bitmap. HI_TDE2_SolidDraw is called as follows:

  ```
  HI_TDE2_SolidDraw(s32Handle,NULL,NULL,pstDst,pstDstRect,
  pstFillColor,pstOpt);
  ```

  The parameter pstOpt can specify the alpha or ROP operation and the clip operation performed on the output result. The colorkey and mirror operations, however, are not supported. When the ROP operation is specified, the operated object S1 indicates the target bitmap pstDst and S2 indicates the fill color.

- When the foreground bitmap is not null (pstOpt cannot be null in this case), HI_TDE2_SolidDraw can be used to perform scaling and anti-flicker operations on the specified area in the foreground bitmap, perform alpha blending or ROP operation on the same specified area and the fill color, and then output the result to the specified area in the target bitmap. HI_TDE2_SolidDraw is called as follows:

  ```
  HI_TDE2_SolidDraw(s32Handle, pstForeGround, pstForeGroundRect,
  pstDst, pstDstRect, pstFillColor, pstOpt);
  ```

  In this case, the fill color can be considered as the background bitmap with the same color. The foreground bitmap and background bitmap support all the operations specified by pstOpt, including the scaling, anti-flicker, and colorkey operations for the foreground bitmap, alpha blending or ROP operation for the foreground bitmap and background bitmap, and mirror and clip operations for the output result.
  - When the ROP operation is specified, the operated object S1 indicates the fill color and S2 indicates the foreground bitmap.
  - When the colorkey operation is specified, only the foreground bitmap supports this operation.
- To draw a rectangle, a vertical line, or a horizontal line by calling HI_TDE2_SolidDraw, you can set the width and height of the filled rectangle. For example, drawing a vertical line is to draw a rectangle with the width of one pixel.
- Hi3518 does not support this API.
- The Hi3520D/Hi3515A/Hi3515C does not support ROP and mirror.

[Example]

None

## HI_TDE2_BitmapMaskRop

[Purpose]

To add the mask ROP operation performed on the raster bitmap to a TDE task. That is, the ROP operation is performed on the foreground bitmap and the background bitmap based on the mask bitmap.

[Syntax]

```
HI_S32 HI_TDE2_BitmapMaskRop(TDE_HANDLE s32Handle,
                             TDE2_SURFACE_S *pstBackGround,
                             TDE2_RECT_S *pstBackGroundRect,
                             TDE2_SURFACE_S *pstForeGround,
                             TDE2_RECT_S *pstForeGroundRect,
                             TDE2_SURFACE_S *pstMask,
                             TDE2_RECT_S *pstMaskRect,
                             TDE2_SURFACE_S *pstDst,
                             TDE2_RECT_S *pstDstRect,
                             TDE_ROP_CODE_E enRopCode_Color,
                             TDE_ROP_CODE_E enRopCode_Alpha);
```

[Description]

The mask bitmap must be an A1 bitmap. In a mask bitmap, the output value of the part indicated by the value 0 is the pixel value of the background bitmap, whereas the output value

of the part indicated by the value 1 is the result obtained after performing the ROP operation on the foreground bitmap and background bitmap.

The differences between the mask ROP operation and a common ROP operation are as follows:

- During a common ROP operation, the ROP operation is performed on all pixel points in the operation areas of the two pictures. That is, the ROP operation cannot be performed on only a part of the operation area (the background cannot be kept).
- A mask ROP operation is implemented through the construction of a proper mask bitmap. Part of the output picture is the ROP result of the foreground bitmap and background bitmap and part of the output picture is the background bitmap. In other words, the ROP result of the foreground bitmap and background bitmap is clipped. After constructing a mask bitmap, you can clip the picture in a random shape.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstBackGround | Background bitmap. | Input |
| pstBackGroundRect | Operation area in the background bitmap. | Input |
| pstForeGround | Foreground bitmap. | Input |
| pstForeGroundRect | Operation area in the foreground bitmap. | Input |
| pstMask | Mask bitmap. | Input |
| pstMaskRect | Operation area in the mask bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| enRopCode_Color | ROP operation code of the color component. | Input |
| enRopCode_Alpha | ROP operation code of the alpha component. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
| --- | --- |
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- When you obtain the valid operation areas between the foreground bitmap, background bitmap, mask bitmap, or target bitmap and their corresponding operation areas, the size of the four valid operation areas must be the same.
- The mask bitmap must be an A1 bitmap.
- The target bitmap and the background bitmap must be in the same color space.
- Hi3518 does not support this API.
- Hi3520D/Hi3515A/Hi3515C does not support this operation.

**NOTE**
The valid operation area refers to the overlapped part of the specified operation area and the bitmap.

[Example]

None

## HI_TDE2_BitmapMaskBlend

[Purpose]

To add the mask blending operation performed on the raster bitmap to a TDE task. That is, the blending operation is performed on the foreground bitmap and the background bitmap with the mask bitmap based on the mask bitmap.

[Syntax]

```
HI_S32 HI_TDE2_BitmapMaskBlend(TDE_HANDLE s32Handle,
                               TDE2_SURFACE_S *pstBackGround,
                               TDE2_RECT_S *pstBackGroundRect,
                               TDE2_SURFACE_S *pstForeGround,
                               TDE2_RECT_S *pstForeGroundRect,
                               TDE2_SURFACE_S *pstMask,
```

```
                                    TDE2_RECT_S *pstMaskRect,
                                    TDE2_SURFACE_S *pstDst,
                                    TDE2_RECT_S *pstDstRect,
                                    HI_U8 u8Alpha,
                                    TDE2_ALUCMD_E enBlendMode);
```

[Description]

The mask bitmap must be an A1 bitmap. In a mask bitmap, the output value of the part indicated by the value 0 is the pixel value of the background bitmap, whereas the output value of the part indicated by the value 1 is the result obtained after performing the blending operation on the foreground bitmap and background bitmap.

The differences between the mask blending operation and a common blending operation are as follows:

- During a common blending operation, the blending operation is performed on all pixel points in the operation areas of the two pictures. That is, the blending operation cannot be performed on only a part of the operation area (the background cannot be kept).
- A mask blending operation is implemented through the construction of a proper mask bitmap. Part of the output picture is the blending result of the foreground bitmap and background bitmap and part of the output picture is the background bitmap. In other words, the blending result of the foreground bitmap and background bitmap is clipped. After constructing a mask bitmap, you can clip the picture in a random shape.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstBackGround | Background bitmap. | Input |
| pstBackGroundRect | Operation area in the background bitmap. | Input |
| pstForeGround | Foreground bitmap. | Input |
| pstForeGroundRect | Operation area in the foreground bitmap. | Input |
| pstMask | Mask bitmap. | Input |
| pstMaskRect | Operation area in the mask bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| u8Alpha | Global alpha value during alpha blending. | Input |
| enBlendMode | Alpha blending mode. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |

| | |
|---|---|
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE and call HI_TDE2_BeginJob to obtain a valid task handle.
- The target bitmap and the background bitmap must be in the same color space.
- The Hi35xx supports the premultiplied mode. If the foreground bitmap is the premultiplied data, select the premultiplied mode for alpha blending; if not, select the non-premultiplied mode.
- The parameter enBlendMode cannot be set to TDE2_ALUCMD_ROP.
- When you obtain the valid operation areas between the foreground bitmap, background bitmap, mask bitmap, or target bitmap and their corresponding operation areas, the size of the four valid operation areas must be the same.
- Hi3518 does not support this API.

[Example]

None

## HI_TDE2_Mb2Mb

[Purpose]

To add the transfer operation with additional functions performed on the macroblock bitmap to a TDE video task. After the format of the operation area in the source macroblock bitmap is converted, the operation area in the source macroblock bitmap is output to the operation area in the target macroblock bitmap. During format conversion, scaling, anti-flicker, and clipping are supported.

[Syntax]

```
HI_S32 HI_TDE2_Mb2Mb(TDE_HANDLE s32Handle,
```

```
 TDE2_MB_S      * pstMBIn,
TDE2_RECT_S   *pstInRect,
 TDE2_MB_S      * pstMbOut,
TDE2_RECT_S   *pstOutRect,
 TDE2_MBOPT_S   * pstMbOpt);
```

[Description]

This API is used to convert the format of the macroblock bitmap. During format conversion, the functions such as scaling, anti-flicker, and clipping are supported for the operation area in the source macroblock bitmap. The scaling mode is specified by setting the **enResize** parameter of **pstMbOpt**. If scaling is set, the operation area in the macroblock bitmap is scaled and then output to the operation area in the target macroblock bitmap. If scaling is not set, the extra part is clipped. If the clip function is enabled, the clip and copy operations are performed.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstMBIn | Input macroblock information. | Input |
| pstInRect | Input macroblock operation rectangle. | Input |
| pstMbOut | Output macroblock information. | Input |
| pstOutRect | Output macroblock operation rectangle. | Input |
| pstMbOpt | Operation attributes of the macroblock bitmap. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |

| Error Code | Description |
| --- | --- |
| HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginVideoJob or HI_TDE2_BeginJob to obtain a valid task handle.
- The minification is less than 255, and there is no limitation on up scaling.
- For the Hi3518, the minification is less than 16, and there is no limitation on up scaling. When the source picture format is 422 and target picture format is 420, the minification is less than 8 in the vertical direction.
- The Hi3518 does not support anti-flicker and clip by calling this API.

[Example]

None

## HI_TDE2_MbFill

[Purpose]

To add a macroblock filling operation to a task.

[Syntax]

```
HI_S32 HI_TDE2_MbFill(TDE_HANDLE s32Handle,
                      TDE2_MB_S *pstMb,
                      TDE2_RECT_S *pstRect,
                      HI_U32 u32YFill,
                      HI_U32 u32CFill,
                      TDE2_MBFILL_E eMbFill);
```

[Description]

This API is used to fill the specified area in a macroblock. You can determine to fill in only the luminance component, fill in only the chrominance component, or fill in both luminance component and chrominance component. When the luminance component is filled, the lower

eight bits of u32YFill are filled; when the chrominance component is filled, the lower 16 bits of u32CFill are filled.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstMb | Macroblock surface. | Input |
| pstRect | Macroblock operation area. | Input |
| u32YFill | Filling value of the luminance component. | Input |
| u32CFill | Filling value of the chrominance component. | Input |
| eMbFill | Macroblock filling mode. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_FAILURE | A system error or an unknown error occurs. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob or HI_TDE2_BeginVideoJob to obtain a valid task handle.
- When the luminance component is filled, the lower eight bits of u32YFill are valid, and the values of the lower eight bits are filled in the chrominance block. When the chrominance component is filled, the lower 16 bits of 32CFill are valid, and the values of the lower 16 bits are filled in the luminance block.
- The chrominance block cannot be filled for the YUV400MB macroblock. Otherwise, an error code is returned, indicating that the operation is not supported.

[Example]

None

## HI_TDE2_Osd2Mb

[Purpose]

To add the operation of blending raster data to the macroblock bitmap to a task. This API is used to perform the alpha blending operation on the raster data and macroblock bitmap. Then the raster bitmap can be blended to the macroblock data in a specified position with a specific transparency.

[Syntax]

```
HI_S32  HI_TDE2_Osd2Mb(TDE_HANDLE s32Handle,
                       TDE2_MB_S* pstBackGround,
                       TDE2_RECT_S  *pstBackGroundRect,
                       TDE2_SURFACE_S* pstForeGround,
                       TDE2_RECT_S  *pstForeGroundRect,
                       TDE2_MB_S* pstMbOut,
                       TDE2_RECT_S  *pstMbOutRect,
                       TDE2_OPT_S* pstOpt);
```

[Description]

The specified area (pstForeGroundRect) of the foreground raster bitmap (pstForeGround) is blended to the specified area (pstForeGroundRect) of the background macroblock bitmap (pstBackGround). Then the blended result is output to the specified area (pstMbOutRect) of the target macroblock bitmap (pstMbOut).

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstBackGround | Background surface (macroblock bitmap). | Input |
| pstBackGroundRect | Operation area in the background. | Input |
| pstForeGround | Foreground surface (raster bitmap). | Input |
| pstForeGroundRect | Operation area in the foreground. | Input |
| pstMbOut | Target surface (macroblock bitmap). | Input |

| Parameter | Description | Input/Output |
|---|---|---|
| pstMbOutRect | Target operation area. | Input |
| pstOpt | Operation parameter settings. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- The supported foreground bitmap formats include ARGB1555, ARGB8888, and ARGB4444. The supported macroblock formats include TDE2_MB_COLOR_FMT_JPG_YCbCr422MBHP (corresponding to PIXEL_FORMAT_YUV_SEMIPLANAR_422 of the VIU) and TDE2_MB_COLOR_FMT_JPG_YCbCr420MBP (corresponding to PIXEL_FORMAT_YUV_SEMIPLANAR_420 of the VIU)
- All the input parameters cannot be set to null; otherwise, error codes are returned.

- A background bitmap and a target macroblock bitmap can be the same. That is, the raster bitmap is blended to the macroblock bitmap in a specified position.
- This API does not support the blending operation after a raster bitmap is scaled. You need to call other APIs of the TDE to scale a raster bitmap, and then call this API to perform the blending operation.
- A bitmap must overlap other operation areas. The overlapped area is the valid operation area in the bitmap. When this API is called, the minimum area among the valid operation areas of the background bitmap, foreground bitmap, and target bitmap is used as the final blending area.
- When a raster bitmap is opaquely blended to a macroblock bitmap with the same size, the raster bitmap is converted into a macroblock bitmap. The transparency during alpha blending is affected by the pixel alpha and global alpha. To implement the opaque effect, note the following two points:
  - Set the operation attribute to alpha blending operation and set the global alpha to opaque as follows:

    stOpt.enAluCmd = TDE2_ALUCMD_BLEND;

    stOpt.u8GlobalAlpha = 0xFF;
  - Set the pixel alpha of the raster bitmap to opaque.

    Take an ARGB8888 bitmap as an example. The upper 8 bits of each pixel indicate the transparency of the pixel, namely, pixel alpha.

    The pixel alpha must be set to 0XFF to implement the opaque effect.

    If the pixel alpha is set to 0x80, you can set the maximum value of the pixel alpha of the surface to 128 (that is, stSuface.bAlphaMax255 = HI_FALSE) to implement the opaque effect of the pixel alpha.
- The operation area size (width x height, in pixels) of the target bitmap cannot be greater than 800x600 pixels.
- The Hi3518 does not support this API.

[Example]

None

## HI_TDE2_CancelJob

[Purpose]

To cancel a TDE task and the added operations.

[Syntax]

```
HI_S32 HI_TDE2_CancelJob(TDE_HANDLE s32Handle);
```

[Description]

When you add an operation to a TDE task, if errors such as invalid operation parameter occur and the program needs to be quit, you can call this API to cancel the TDE task and all operations.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_FAILURE | The specified task has been submitted and cannot be canceled. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- Before calling this API, call HI_TDE2_Open to start the TDE device and call HI_TDE2_BeginJob to obtain a valid task handle.
- A submitted task cannot be canceled.
- A task becomes invalid after it is canceled. Therefore, no operation can be added to the task and the task cannot be submitted.
- If an error occurs when you add an operation (such as operation A) to a TDE task, you can process such problem by using either of the following methods:
  - Ignore operation A and add other operations to the TDE task, and then submit the task. If the task is run successfully, it indicates that all operations that are successfully added are finished, and operation A is not performed because it fails to be added.
  - Cancel the task. Then all successfully added operations of the TDE task are canceled.

[Example]

```
/* declaration */
    HI_S32  s32Ret;
    TDE_HANDLE s32Handle;
    TDE2_SURFACE_S stSrc;
    TDE2_SURFACE_S stDst;
    TDE2_OPT_S stOpt = {0};

    /* create a TDE job */
    s32Handle = HI_API_TDE_BeginJob();
    if(HI_ERR_TDE_INVALID_HANDLE == s32Handle)
```

```
    {
        return -1;
    }

    /* add several commands to job successfully*/
...

    /* prepare arguments of bitblit command */

    /* if fail to add one more bitblt command to the job, cancel the job*/
    s32Ret = HI_API_TDE_BitBlt(s32Handle, &stSrc, &stDst, &stOpt);
    if(HI_SUCCESS != s32Ret)
    {
        printf("add bitlit command failed!\n");
       HI_TDE2_CancleJob(s32Handle);
       return -1;
    }
```

## HI_TDE2_WaitForDone

[Purpose]

To wait for the completion of a specific TDE task.

[Syntax]

```
HI_S32 HI_TDE2_WaitForDone(TDE_HANDLE s32Handle);
```

[Description]

When you submit a TDE task in non-block mode, you can call this API to wait the completion of the task. This API is called in block mode.

After the TDE performs an asynchronous (non-block mode) operation on a display buffer, the software performs operations on the display buffer. This may result in the risk of performing operations on the same display buffer concurrently by the TDE and software. At this time, you can call this API to ensure that the TDE task is finished, and then perform operations through software.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | The specified TDE task is complete. |

| | |
|---|---|
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_QUERY_TIMEOUT | The specific task is not complete due to timeout. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- As a block interface, this API is used to block the task of waiting for the completion of a specified task.
- It is prohibited to wait for an unsubmitted task; otherwise, the error code HI_ERR_TDE_UNSUPPORTED_OPERATION is returned.

[Example]

None

## HI_TDE2_MbQuickCopy

[Purpose]

To add an operation of rapidly copying macroblock bitmaps to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_MbQuickCopy(TDE_HANDLE s32Handle,
                           TDE2_MB_S *pstSrc,
                           TDE2_RECT_S *pstSrcRect,
                           TDE2_MB_S *pstDst,
                           TDE2_RECT_S *pstDstRect);
```

[Description]

This API is used to copy a specified area (pstSrcRect) of the bitmap with the base address pstSrc to the memory whose destination address is pstDst and output area is pstDstRect.

For details about the bitmap, operation area, and the relationship between them, see the descriptions of HI_TDE2_QuickCopy.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | The specified TDE task is complete. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- The function of HI_TDE2_MbQuickCopy is implemented by using DMA transfer; therefore, HI_TDE2_MbQuickCopy is superior to HI_TDE2_Mb2Mb in transfer.
- As format conversion is not supported during the rapid copy operation, the source bitmap format must be the same as the destination bitmap format.

- Scaling is not supported during the rapid copy operation. If the operation area sizes of the source bitmap and target bitmap are different, the copy and transfer operations are performed based on the minimum public area of the source bitmap and target bitmap.
- This API applies only to the Hi3518.

[Example]

None

## HI_TDE2_MbRotate

[Purpose]

To add an operation of rotating macroblock bitmaps to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_MbRotate(TDE_HANDLE s32Handle,
                        TDE2_MB_S *pstSrc,
                        TDE2_RECT_S *pstSrcRect,
                        TDE2_MB_S *pstDst,
                        TDE2_RECT_S *pstDstRect,
                        TDE_ROTATE_ANGLE_E enRotateAngle);
```

[Description]

This API is used to clockwise rotate a specified area (pstSrcRect) of the bitmap with the base address pstSrc by a specified angle, and then output the rotated area to the memory whose destination address is pstDst and output area is pstDstRect.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| enRotateAngle | Rotation angle. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | The specified TDE task is complete. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

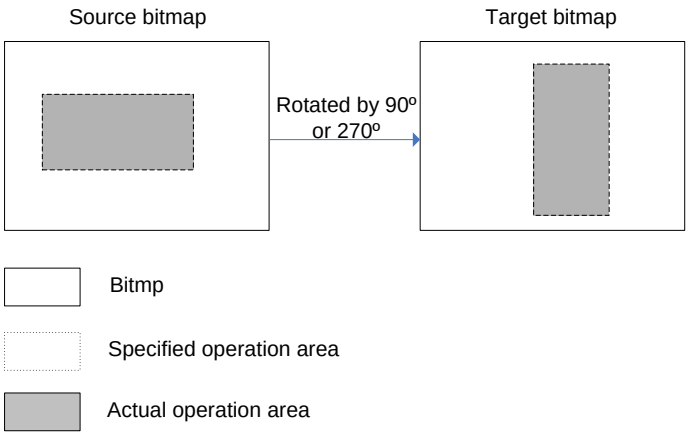
[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

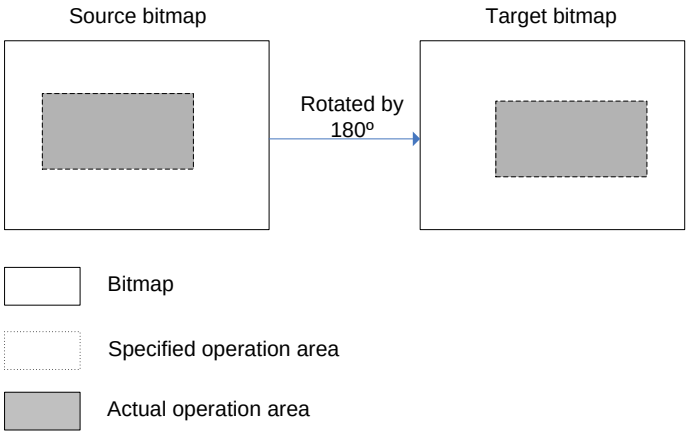
- This API applies only to the Hi3518.
- Only the pictures in 420 format are supported.
- The following are limitations on the addresses, operation areas, and strides of the source bitmap and target bitmap:
  - The start addresses of the source bitmap and target bitmap must be 16-byte-aligned.
  - The horizontal start position must be 16-byte-aligned if the operation area is rotated by 90º.
  - The horizontal end position must be 16-byte-aligned if the operation area is rotated by 180º or 270º.
  - The strides of the source bitmap and target bitmap must be 16-byte-aligned.
- The following describes the requirements on the operation areas in the source bitmap and target bitmap:
  - If the operation area is rotated by 90º or 270º, the width and height of the operation area in the source bitmap are the same as the height and width of the operation area in the target bitmap respectively, as shown in Figure 2-8.

**Figure 2-8** Matched operation areas in the source bitmap and target bitmap when the operation area is rotated by 90º or 270º




- If the operation area is rotated by 180º, the width and height of the operation area in the source bitmap are the same as the width and height of the operation area in the target bitmap respectively, as shown in Figure 2-9.

**Figure 2-9** Matched operation areas in the source bitmap and target bitmap when the operation area is rotated by 180º




- If the widths and heights of the operation areas in the source bitmap and target bitmap do not match, the operation areas are clipped. This ensures that the widths and heights of the operation areas match. See Figure 2-10 and Figure 2-11.

**Figure 2-10** Unmatched operation areas in the source bitmap and target bitmap when the operation area is rotated by 90º or 270º

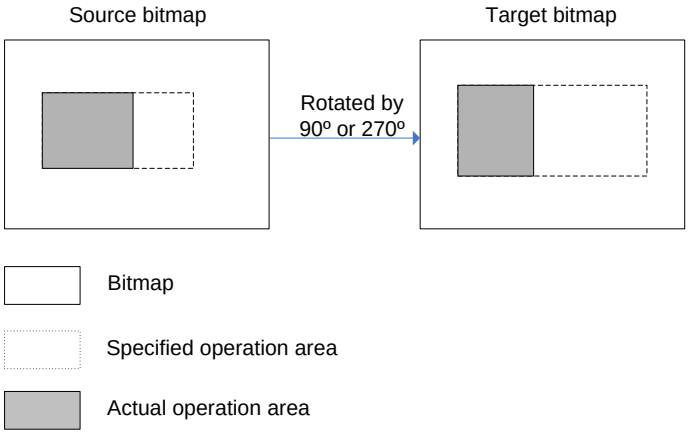
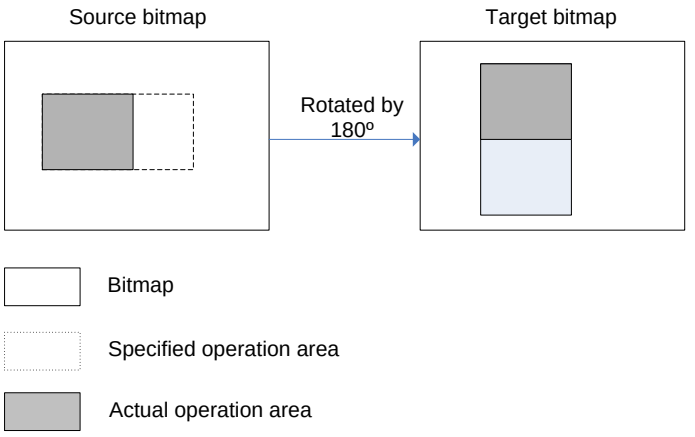



**Figure 2-11** Unmatched operation areas in the source bitmap and target bitmap when the operation area is rotated by 180º




[Example]

None

## HI_TDE2_MbLDC

[Purpose]

To add an operation of vertically correcting distorted macroblock bitmaps to a TDE task.

[Syntax]

```
HI_S32 HI_TDE2_MbLDC(TDE_HANDLE s32Handle,
                     TDE2_MB_S *pstSrc,
                     TDE2_RECT_S *pstSrcRect,
                     TDE2_MB_S *pstDst,
                     TDE2_RECT_S *pstDstRect,
```

```
                                LDC_ATTR_S *pstLDCAttr);
```

[Description]

The pictures captured by some low-end lenses are easily distorted. Such pictures need to corrected.

This API is used to correct a specified area (pstSrcRect) of the bitmap with the base address pstSrc, and then output the corrected area to the memory whose destination address is pstDst and output area is pstDstRect.

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| s32Handle | TDE task handle. | Input |
| pstSrc | Source bitmap. | Input |
| pstSrcRect | Operation area in the source bitmap. | Input |
| pstDst | Target bitmap. | Input |
| pstDstRect | Operation area in the target bitmap. | Input |
| pstLDCAttr | Distortion correction attribute. | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | The specified TDE task is complete. |
| Other values | Failure. Its value is an error code. |

[Error Code]

| Error Code | Description |
|---|---|
| HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| HI_FAILURE | A system error or an unknown error occurs. |

[Requirement]

- Header file: hi_tde_api.h
- Library file: libtde.a

[Note]

- This API applies only to the Hi3518.
- Only the pictures in 420 formats are supported.
- The following are limitations on the addresses, operation areas, and strides of the source bitmap and target bitmap:
  - The start addresses of the source bitmap and target bitmap must be 16-byte-aligned.
  - The start positions of the operation areas in the source bitmap and target bitmap are (0, 0).
  - The strides of the source bitmap and target bitmap must be 16-byte-aligned.

[Example]

None

# 3 Data Types

## 3.1 Mapping Table

This chapter describes the data types related to APIs, as shown in Table 3-1.

**Table 3-1** TDE data types

| Data Type | Description |
|---|---|
| TDE_HANDLE | Defines the TDE task handle. |
| TDE2_COLOR_FMT_E | Defines the raster pixel format supported by the TDE. |
| TDE2_RECT_S | Defines the attributes of the operation area. |
| TDE2_ALUCMD_E | Defines the TDE logical operation type. |
| TDE_ROP_CODE_E | Defines the ROP type supported by the TDE. |
| TDE2_COLORKEY_MODE_E | Defines the colorkey mode. |
| TDE2_COLORKEY_COMP_S | Defines the colorkey attributes of each color component. |
| TDE2_COLORKEY_U | Defines the attributes of the colorkey. |
| TDE2_CLIPMODE_E | Defines the clip mode. |
| TDE2_OUTALPHA_FROM_E | Defines the type of the alpha output source. |
| TDE2_FILTER_MODE_E | Defines the picture filtering mode. |
| TDE2_MIRROR_E | Defines the mirror attributes of a picture. |
| TDE2_SURFACE_S | Defines the surface of a bitmap. |
| TDE2_OPT_S | Defines the attributes of a TDE operation. |
| TDE2_MB_COLOR_FMT_E | Defines the macroblock format supported by the TDE. |
| TDE_PIC_MODE_E | Defines the processing mode of the frame or field. |

| Data Type | Description |
| --- | --- |
| TDE2_MB_S | Defines the basic attributes of a macroblock bitmap. |

# 3.2 Description of Data Types

## TDE_HANDLE

[Description]

Defines the TDE task handle.

[Syntax]

```
typedef HI_S32 TDE_HANDLE;
```

[Member]

None

[Note]

None

[See Also]

None

## TDE2_COLOR_FMT_E

[Description]

Defines the pixel format supported by the TDE.

[Syntax]

```
typedef enum hiTDE2_COLOR_FMT_E
{
    TDE2_COLOR_FMT_RGB444 = 0,
    TDE2_COLOR_FMT_RGB555,
    TDE2_COLOR_FMT_RGB565,
    TDE2_COLOR_FMT_RGB888,
    TDE2_COLOR_FMT_ARGB4444,
    TDE2_COLOR_FMT_ARGB1555,
    TDE2_COLOR_FMT_ARGB8565,
    TDE2_COLOR_FMT_ARGB8888,
    TDE2_COLOR_FMT_CLUT1,
    TDE2_COLOR_FMT_CLUT2,
    TDE2_COLOR_FMT_CLUT4,
    TDE2_COLOR_FMT_CLUT8,
```

```
    TDE2_COLOR_FMT_ACLUT44,
    TDE2_COLOR_FMT_ACLUT88,
    TDE2_COLOR_FMT_A1,
    TDE2_COLOR_FMT_A8,
    TDE2_COLOR_FMT_YCbCr888,
    TDE2_COLOR_FMT_AYCbCr8888,
    TDE2_COLOR_FMT_YCbCr422,
    TDE2_COLOR_FMT_byte,
    TDE2_COLOR_FMT_halfword,
    TDE2_COLOR_FMT_BUTT
} TDE2_COLOR_FMT_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_COLOR_FMT_RGB444 | RGB444 format |
| TDE2_COLOR_FMT_RGB555 | RGB555 format |
| TDE2_COLOR_FMT_RGB565 | RGB565 format |
| TDE2_COLOR_FMT_RGB888 | RGB888 format |
| TDE2_COLOR_FMT_ARGB4444 | ARGB4444 format |
| TDE2_COLOR_FMT_ARGB1555 | ARGB1555 format |
| TDE2_COLOR_FMT_ARGB8565 | ARGB8565 format |
| TDE2_COLOR_FMT_ARGB8888 | ARGB8888 format |
| TDE2_COLOR_FMT_CLUT1 | CLUT1 format |
| TDE2_COLOR_FMT_CLUT4 | CLUT4 format |
| TDE2_COLOR_FMT_CLUT8 | CLUT8 format |
| TDE2_COLOR_FMT_ACLUT44 | ACLUT44 format |
| TDE2_COLOR_FMT_ACLUT88 | ACLUT88 format |
| TDE2_COLOR_FMT_A1 | A1 format |
| TDE2_COLOR_FMT_A8 | A8 format |
| TDE2_COLOR_FMT_YCbCr888 | YCbCr888 format |
| TDE2_COLOR_FMT_AYCbCr8888 | AYCbCr8888 format |
| TDE2_COLOR_FMT_YCbCr422 | YCbCr422 format |
| TDE2_COLOR_FMT_byte | Byte format |
| TDE2_COLOR_FMT_halfword | Halfword format |
| TDE_COLOR_FMT_BUTT | Invalid |

[Note]

- The Hi3518 supports only TDE2_COLOR_FMT_ARGB4444 and TDE2_COLOR_FMT_ARGB1555 formats.
- The Hi3520D/Hi3515A/Hi3515C supports only the following formats: ARGB4444, ARGB1555, ARGB8888, YCbCr422, byte, and halfword.

[See Also]

None

## TDE2_RECT_S

[Description]

Defines the attributes of the operation area of the TDE.

[Syntax]

```
typedef struct hiTDE2_RECT_S
{
    HI_S32 s32Xpos;
    HI_S32 s32Ypos;
    HI_U32 u32Width;
    HI_U32 u32Height;
} TDE2_RECT_S;
```

[Member]

| Member | Description |
|---|---|
| s32Xpos | Start horizontal coordinate of the operation area (in pixels).<br>Valid range: [0, bitmap width) |
| s32Ypos | Start vertical coordinate of the operation area (in pixels).<br>Valid range: [0, bitmap height) |
| u32Width | Width of the operation area (in pixels).<br>Valid range: (0, 0xFFF] |
| u32Height | Height of the operation area (in pixels).<br>Valid range: (0, 0xFFF] |

[Note]

- For the details about the relationships between bitmaps and operation areas, see Figure 2-1.
- If an operation area overlaps a bitmap, the overlapped area is served as the actual operation area; if an operation area does not overlap a bitmap, an error code is returned.

[See Also]

None

## TDE2_ALUCMD_E

[Description]

Defines the attributes of the logical operation types.

[Syntax]

```
typedef enum hiTDE2_ALUCMD_E
{
    TDE2_ALUCMD_NONE = 0,
    TDE2_ALUCMD_BLEND,
    TDE2_ALUCMD_ROP,
    TDE2_ALUCMD_COLORIZE,
    TDE2_ALUCMD_BUTT
} TDE2_ALUCMD_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_ALUCMD_NONE | No logical operation. |
| TDE2_ALUCMD_BLEND | Alpha blending type. |
| TDE2_ALUCMD_ROP | Boolean operation type. |
| TDE2_ALUCMD_COLORIZE | Colorize operation. |
| TDE2_ALUCMD_BUTT | Invalid |

[Note]

- To perform the alpha blending operation on two bitmaps, select TDE2_ALUCMD_BLEND; to perform the colorize operation, select TDE2_ALUCMD_COLORIZE.
- If TDE2_ALUCMD_ROP is selected, the boolean logical operation is performed. You can specify the members enRopCode_Color and enRopCode_Alpha of TDE2_OPT_S to specify the ROP types of the color component and the alpha component.

[See Also]

None

## TDE_ROP_CODE_E

[Description]

Defines the ROP type supported by the TDE.

[Syntax]

```
typedef enum hiTDE2_ROP_CODE_E
```

```
{
    TDE2_ROP_BLACK = 0,         /*Blackness*/
    TDE2_ROP_NOTMERGEPEN,       /*~(S2+S1)*/
    TDE2_ROP_MASKNOTPEN,        /*~S2&S1*/
    TDE2_ROP_NOTCOPYPEN,        /* ~S2*/
    TDE2_ROP_MASKPENNOT,        /* S2&~S1 */
    TDE2_ROP_NOT,               /* ~S1 */
    TDE2_ROP_XORPEN,            /* S2^S1 */
    TDE2_ROP_NOTMASKPEN,        /* ~(S2&S1) */
    TDE2_ROP_MASKPEN,           /* S2&S1 */
    TDE2_ROP_NOTXORPEN,         /* ~(S2^S1) */
    TDE2_ROP_NOP,               /* S1 */
    TDE2_ROP_MERGENOTPEN,       /* ~S2+S1 */
    TDE2_ROP_COPYPEN,           /* S2 */
    TDE2_ROP_MERGEPENNOT,       /* S2+~S1 */
    TDE2_ROP_MERGEPEN,          /* S2+S1 */
    TDE2_ROP_WHITE,             /* Whiteness */
    TDE2_ROP_BUTT
} TDE2_ROP_CODE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_ROP_BLACK | Blackness |
| TDE2_ROP_NOTMERGEPEN | ~(S2+S1) |
| TDE2_ROP_MASKNOTPEN | ~S2&S1 |
| TDE2_ROP_NOTCOPYPEN | ~S2 |
| TDE2_ROP_MASKPENNOT | S2&~S1 |
| TDE2_ROP_NOT | ~S1 |
| TDE2_ROP_XORPEN | S2^S1 |
| TDE2_ROP_NOTMASKPEN | ~(S2&S1) |
| TDE2_ROP_MASKPEN | S2&S1 |
| TDE2_ROP_NOTXORPEN | ~(S2^S1) |
| TDE2_ROP_NOP | S1 |
| TDE2_ROP_MERGENOTPEN | ~S2+S1 |
| TDE2_ROP_COPYPE | S2 |
| TDE2_ROP_MERGEPENNOT | S2+~S1 |
| TDE2_ROP_MERGEPEN | S2+S1 |

| Member | Description |
| --- | --- |
| TDE2_ROP_WHITE | Whiteness |
| TDE_ROP_BUTT | Invalid |

**NOTE**
S1 indicates bitmap 1 and S2 indicates bitmap 2.

[Note]

The bitmaps indicated by S1 and S2 vary according to operations. For details, see the description of each API. If the operation type for two bitmaps is set to TDE2_ALUCMD_ROP, different ROP operations can be specified for the color space and alpha. Assume that the foreground bitmap and background bitmap are in ARGB8888 format, the pixel value of the foreground bitmap is foreground, the pixel value of the background bitmap is background, the pixel value after operation is pixel, the ROP operation for alpha is whiteness, and the ROP operation for the color space is blackness. Then the pixel values after operation are as follows:

- pixel.alpha = 0xff
- pixel.r = pixel.g = pixel.b = 0x00

where **pixel.alpha**, **pixel.r**, **pixel.g**, and **pixel.b** indicate the bitmap components after operation.

[See Also]

None

## TDE2_COLORKEY_MODE_E

[Description]

Defines the attributes of the colorkey mode of the TDE.

[Syntax]

```
typedef enum hiTDE2_COLORKEY_MODE_E
{
    TDE2_COLORKEY_MODE_NONE = 0,
    TDE2_COLORKEY_MODE_FOREGROUND,
    TDE2_COLORKEY_MODE_BACKGROUND,
    TDE2_COLORKEY_MODE_BUTT
} TDE2_COLORKEY_MODE_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE2_COLORKEY_MODE_NONE | Does not perform the colorkey operation. |
| TDE2_COLORKEY_MODE_FOREGROUND | Performs the colorkey operation on the foreground bitmap. |

| Member | Description |
| --- | --- |
| TDE2_COLORKEY_MODE_BACKGROUND | Performs the colorkey operation on the background bitmap. |
| TDE2_COLORKEY_MODE_BUTT | Invalid |

[Note]

When performing the colorkey operation on the foreground bitmap, the TDE performs this operation before the CLUT for color extension and performs this operation after the CLUT for color correction.

[See Also]

None

## TDE2_COLORKEY_COMP_S

[Description]

Defines the colorkey attributes of each color component.

[Syntax]

```
typedef struct hiTDE2_COLORKEY_COMP_S
{
    HI_U8 u8CompMin;          /*Minimum colorkey of a component.*/
    HI_U8 u8CompMax;          /*Maximum colorkey of a component.*/
    HI_U8 bCompOut;           /*The colorkey of a component is within or out
of the range.*/
    HI_U8 bCompIgnore;       /*Whether to ignore a component.*/

     HI_U8 u8CompMask;         /**<Component mask*/

    HI_U8 u8Reserved;

    HI_U8 u8Reserved1;

    HI_U8 u8Reserved2;
} TDE2_COLORKEY_COMP_S;
```

[Member]

| Member | Description |
| --- | --- |
| u8CompMin | Minimum colorkey of a component. |
| u8CompMax | Maximum colorkey of a component. |
| bCompOut | The colorkey of a component is within or out of the range. |
| bCompIgnore | Whether to ignore a component. |

| Member | Description |
|---|---|
| u8CompMask | Component mask. This parameter is meaningless, and its value range is 0–255. |
| u8Reserved–u8Reserved2 | Reserved. |

[Note]

The member bCompIgnore specifies whether to ignore a component during colorkey comparison and considers that the component always meets the colorkey requirements.

- If bCompIgnore is set to TRUE, it indicates that a component is ignored during colorkey comparison and it is considered that the component always meets the colorkey requirements.
- If bCompIgnore is set to FALSE, it indicates that the TDE checks whether the component meets the colorkey requirements based on the [minimum colorkey, maximum colorkey] and the value of bCompOut.

[See Also]

None

## TDE2_COLORKEY_U

[Description]

Defines the attributes of the colorkey.

[Syntax]

```
typedef union hiTDE2_COLORKEY_U
{
    struct
    {
        TDE2_COLORKEY_COMP_S stAlpha;
        TDE2_COLORKEY_COMP_S stRed;
        TDE2_COLORKEY_COMP_S stGreen;
        TDE2_COLORKEY_COMP_S stBlue;
    } struCkARGB;
    struct
    {
        TDE2_COLORKEY_COMP_S stAlpha;
        TDE2_COLORKEY_COMP_S stY;
        TDE2_COLORKEY_COMP_S stCb;
        TDE2_COLORKEY_COMP_S stCr;
    } struCkYCbCr;
    struct
    {
        TDE2_COLORKEY_COMP_S stAlpha;
```

```
        TDE2_COLORKEY_COMP_S stClut;
    } struCkClut;
} TDE2_COLORKEY_U;
```

[Member]

The member struCkARGB indicates the colorkey attributes of each component when the bitmap is in the ARGB format.

| Member | Description |
|---|---|
| stAlpha | Colorkey attributes of the alpha component |
| stRed | Colorkey attributes of the R component |
| stGreen | Colorkey attributes of the G component |
| stBlue | Colorkey attributes of the B component |

The member struCkYCbCr indicates the colorkey attributes of each component when the bitmap is in the AYCbCr format.

| Member | Description |
|---|---|
| stAlpha | Colorkey attributes of the alpha component |
| stY | Colorkey attributes of the Y component |
| stCb | Colorkey attributes of the Cb component |
| stCr | Colorkey attributes of the Cr component |

The member struCkClut indicates the colorkey attributes of each component when the bitmap is in the CLUT format.

| Member | Description |
|---|---|
| stAlpha | Colorkey attributes of the alpha component |
| stClut | Colorkey attributes of the CLUT component |

The member TDE2_COLORKEY_U indicates the colorkey attributes of each component.

| Member | Description |
|---|---|
| struCkARGB | Colorkey attributes when the bitmap is in the ARGB format |
| struCkYCbCr | Colorkey attributes when the bitmap is in the AYCbCr format |
| struCkClut | Colorkey attributes when the bitmap is in the CLUT format |

[Note]

- Regardless of the format of the current bitmap, the maximum and minimum values of the color space must be in ARGB8888 format.
- Because the Hi3518 supports only the monochromatic colorkey, the color upper limit and lower limit must be the same.

[See Also]

None

## TDE2_CLIPMODE_E

[Description]

Defines the clip mode.

[Syntax]

```
typedef enum hiTDE2_CLIPMODE_E
{
    TDE2_CLIPMODE_NONE = 0,
    TDE2_CLIPMODE_INSIDE,
    TDE2_CLIPMODE_OUTSIDE,
    TDE2_CLIPMODE_BUTT
} TDE2_CLIPMODE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_CLIPMODE_NONE | No clip for the output result |
| TDE2_CLIPMODE_INSIDE | Intra-area clip mode |
| TDE2_CLIPMODE_OUTSIDE | Extra-area clip mode |
| TDE2_CLIPMODE_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_OUTALPHA_FROM_E

[Description]

Defines the type of the alpha output source.

[Syntax]

```
typedef enum hiTDE2_OUTALPHA_FROM_E
```

```
{
    TDE2_OUTALPHA_FROM_NORM = 0,
    TDE2_OUTALPHA_FROM_BACKGROUND,
    TDE2_OUTALPHA_FROM_FOREGROUND,
    TDE2_OUTALPHA_FROM_GLOBALALPHA,
    TDE2_OUTALPHA_FROM_BUTT
} TDE2_OUTALPHA_FROM_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE2_OUTALPHA_FROM_NORM | The alpha value of the output picture is derived from the result of the alpha blending or anti-flicker operation. |
| TDE2_OUTALPHA_FROM_BACKGROUND | The alpha value of the output picture is derived from the background bitmap. |
| TDE2_OUTALPHA_FROM_FOREGROUND | The alpha value of the output picture is derived from the foreground bitmap. |
| TDE2_OUTALPHA_FROM_GLOBALALPHA | The alpha value of the output picture is derived from the global alpha value. |

[Note]

None

[See Also]

None

## TDE2_DEFLICKER_MODE_E

[Description]

Defines the anti-flicker configuration of a channel.

[Syntax]

```
typedef enum hiTDE2_DEFLICKER_MODE_E
{
    TDE2_DEFLICKER_MODE_NONE = 0,
    TDE2_DEFLICKER_MODE_RGB,
    TDE2_DEFLICKER_MODE_BOTH,
    TDE2_DEFLICKER_MODE_BUTT
}TDE2_DEFLICKER_MODE_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE2_DEFLICKER_MODE_NONE | No anti-flicker |
| TDE2_DEFLICKER_MODE_RGB | Anti-flicker on RGB component |
| TDE2_DEFLICKER_MODE_BOTH | Anti-flicker on alpha component |
| TDE2_DEFLICKER_MODE_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_DEFLICKER_LEVEL_E

[Description]

Defines the anti-flicker level.

[Syntax]

```
typedef enum hiTDE_DEFLICKER_LEVEL_E
{
    TDE_DEFLICKER_AUTO = 0,
    TDE_DEFLICKER_LOW,
    TDE_DEFLICKER_MIDDLE,
    TDE_DEFLICKER_HIGH,
    TDE_DEFLICKER_BUTT
}TDE_DEFLICKER_LEVEL_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE_DEFLICKER_AUTO | Adaptation. The anti-flicker coefficient is selected by the TDE. |
| TDE_DEFLICKER_LOW | Low-level anti-flicker |
| TDE_DEFLICKER_MIDDLE | Medium-level anti-flicker |
| TDE_DEFLICKER_HIGH | Intermediate-level anti-flicker |
| TDE_DEFLICKER_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_BLEND_MODE_E

[Description]

Defines the user-defined alpha blending mode.

[Syntax]

```
typedef enum hiTDE2_BLEND_MODE_E
{
    TDE2_BLEND_ZERO = 0x0,
    TDE2_BLEND_ONE,
    TDE2_BLEND_SRC2COLOR,
    TDE2_BLEND_INVSRC2COLOR,
    TDE2_BLEND_SRC2ALPHA,
    TDE2_BLEND_INVSRC2ALPHA,
    TDE2_BLEND_SRC1COLOR,
    TDE2_BLEND_INVSRC1COLOR,
    TDE2_BLEND_SRC1ALPHA,
    TDE2_BLEND_INVSRC1ALPHA,
    TDE2_BLEND_SRC2ALPHASAT,
    TDE2_BLEND_BUTT
}TDE2_BLEND_MODE_E;
```

[Member]

Pixel = (Foreground x fs + Background x fd)

where

- fs: foreground blend coefficient
- fd: destination blend coefficient
- Pixel: pixel value after operation
- Foreground: pixel value of the foreground bitmap
- Background: pixel value of the background bitmap
- sa: foreground alpha
- da: background alpha
- sc: foreground color
- dc: background color

- fs and fd: pixel coefficients of the source bitmap and target bitmap respectively. Each member indicates a coefficient.

| Member | Description |
|---|---|
| TDE2_BLEND_ZERO | 0 |
| TDE2_BLEND_ONE | 1 |
| TDE2_BLEND_SRC2COLOR | sc |
| TDE2_BLEND INVSRC2COLOR | 1 – sc |
| TDE2_BLEND SRC2ALPHA | sa |
| TDE2_BLEND INVSRC2ALPHA | 1 – sa |
| TDE2_BLEND SRC1COLOR | dc |
| TDE2_BLEND INVSRC1COLOR | 1 – dc |
| TDE2_BLEND SRC1ALPHA | da |
| TDE2_BLEND INVSRC1ALPHA | 1-da |
| TDE2_BLEND SRC2ALPHASAT | min(1 – da, sa) + 1 |
| TDE2_BLEND_BUTT | Invalid |

[Note]

When alpha blending is performed on the foreground bitmap and background bitmap, the blending mode of the Src1 channel and Src2 channel can be independently set. Currently, eleven blending modes are supported. When TDE2_BLENDCMD_E is set to TDE2_BLENDCMD_CONFIG, you can select the blending mode by setting TDE2_BLEND_MODE_E.

[See Also]

None

## TDE2_BLENDCMD_E

[Description]

Defines the alpha blending command. This command is used to calculate the pixel value after alpha blending.

[Syntax]

```
typedef  enum  hiTDE2_BLENDCMD_E
{
    TDE2_BLENDCMD_NONE = 0x0,
    TDE2_BLENDCMD_CLEAR,
    TDE2_BLENDCMD_SRC,
```

```
    TDE2_BLENDCMD_SRCOVER,
    TDE2_BLENDCMD_DSTOVER,
     TDE2_BLENDCMD_SRCIN,
     TDE2_BLENDCMD_DSTIN,
    TDE2_BLENDCMD_SRCOUT,
    TDE2_BLENDCMD_DSTOUT,
    TDE2_BLENDCMD_SRCATOP,
    TDE2_BLENDCMD_DSTATOP,
    TDE2_BLENDCMD_ADD,
    TDE2_BLENDCMD_XOR,
    TDE2_BLENDCMD_DST,
    TDE2_BLENDCMD_CONFIG,
    TDE2_BLENDCMD_BUTT
}TDE2_BLENDCMD_E
```

[Member]

Pixel = (Foreground x fs + Background x fd)

where,

- fs: foreground blend coefficient
- fd: destination blend coefficient
- Pixel: pixel value after operation
- Foreground: pixel value of the foreground bitmap
- Background: pixel value of the background bitmap
- sa: foreground alpha
- da: background alpha

| Member | Description |
|---|---|
| TDE2_BLENDCMD_NONE | fs is valued at sa and fd is valued at (1.0 – sa). |
| TDE2_BLENDCMD_CLEAR | Both fs and fd are valued at 0.0. |
| TDE2_BLENDCMD_SRC | fs is valued at 1.0 and fd is valued at 0.0. |
| TDE2_BLENDCMD_SRCOVER | fs is valued at 1.0 and fd is valued at (1.0 – sa). |
| TDE2_BLENDCMD_DSTOVER | fs is valued at (1.0 – da) and fd is valued at 1.0. |
| TDE2_BLENDCMD_SRCIN | fs is valued at da and fd is valued at 0.0. |
| TDE2_BLENDCMD_DSTIN | fs is valued at 0.0 and fd is valued at sa. |
| TDE2_BLENDCMD_SRCOUT | fs is valued at (1.0 – da) and fd is valued at 0.0. |
| TDE2_BLENDCMD_DSTOUT | fs is valued at 0.0 and fd is valued at (1.0 – sa). |

| Member | Description |
|---|---|
| TDE2_BLENDCMD_SRCATOP | fs is valued at da and fd is valued at (1.0 – sa). |
| TDE2_BLENDCMD_DSTATOP | fs is valued at (1.0 – da) and fd is valued at sa. |
| TDE2_BLENDCMD_ADD | Both fs and fd are valued at 1.0. |
| TDE2_BLENDCMD_XOR | fs is valued at (1.0 – da) and fd is valued at (1.0 – sa). |
| TDE2_BLENDCMD_DST | fs is valued at 0.0 and fd is valued at 1.0. |
| TDE2_BLENDCMD_CONFIG | User-defined configuration |
| TDE2_BLENDCMD_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_BLEND_OPT_S

[Description]

Defines the options for alpha blending.

[Syntax]

```
typedef struct hiTDE2_BLEND_OPT_S
{
    HI_BOOL  bGlobalAlphaEnable;
    HI_BOOL  bPixelAlphaEnable;
    HI_BOOL bSrc1AlphaPremulti;
    HI_BOOL bSrc2AlphaPremulti;
    TDE2_BLENDCMD_E eBlendCmd;
    TDE2_BLEND_MODE_E eSrc1BlendMode;
    TDE2_BLEND_MODE_E eSrc2BlendMode;
}TDE2_BLEND_OPT_S;
```

[Member]

| Member | Description |
|---|---|
| bGlobalAlphaEnable | Global alpha enable. |
| bPixelAlphaEnable | Pixel alpha enable. |

| Member | Description |
| --- | --- |
| bSrc1AlphaPremulti | Src1 alpha premultiply enable. |
| bSrc2AlphaPremulti | Src2 alpha premultiply enable. |
| eBlendCmd | Alpha blending command. |
| eSrc1BlendMode | Src1 blending mode select. It is valid when eBlendCmd is set to TDE2_BLENDCMD_CONFIG. |
| eSrc2BlendMode | Src2 blending mode select. It is valid when eBlendCmd is set to TDE2_BLENDCMD_CONFIG. |

[Note]

None

[See Also]

None

## TDE2_PATTERN_FILL_OPT_S

[Description]

Defines the information about pattern filling.

[Syntax]

```
typedef struct hiTDE2_PATTERN_FILL_OPT_S
{
    TDE2_ALUCMD_E enAluCmd;
    TDE2_ROP_CODE_E enRopCode_Color;
    TDE2_ROP_CODE_E enRopCode_Alpha;
    TDE2_COLORKEY_MODE_E enColorKeyMode;
    TDE2_COLORKEY_U unColorKeyValue;
    TDE2_CLIPMODE_E enClipMode;
    TDE2_RECT_S stClipRect;
    HI_BOOL bClutReload;
    HI_U8 u8GlobalAlpha;
    TDE2_OUTALPHA_FROM_E enOutAlphaFrom;
    HI_U32 u32Colorize;
    TDE2_BLEND_OPT_S stBlendOpt;
    TDE2_CSC_OPT_S stCscOpt;
}TDE2_PATTERN_FILL_OPT_S;
```

[Member]

| Member | Description |
| --- | --- |
| enAluCmd | Logical operation type |

| Member | Description |
|---|---|
| enRopCode_Color | ROP type of the color space |
| enRopCode_Alpha | ROP type of the alpha |
| enColorKeyMode | Colorkey mode |
| unColorKeyValue | Colorkey value |
| enClipMode | Clip mode |
| stClipRect | Clipped area |
| bClutReload | Whether to reload the CLUT |
| u8GlobalAlpha | Global alpha |
| enOutAlphaFrom | Alpha output source |
| u32Colorize | Colorize value |
| stBlendOpt | Blending option |
| stCscOpt | CSC parameter option |

[Note]

None

[See Also]

None

## TDE2_FILTER_MODE_E

[Description]

Defines the picture filtering mode.

[Syntax]

```
typedef enum hiTDE2_FILTER_MODE_E
{
    TDE2_FILTER_MODE_COLOR = 0,
    TDE2_FILTER_MODE_ALPHA,
    TDE2_FILTER_MODE_BOTH,
    TDE2_FILTER_MODE_BUTT
} TDE2_FILTER_MODE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_FILTER_MODE_COLOR | Filters the color. |
| TDE2_FILTER_MODE_ALPHA | Filters the alpha channel. |

| Member | Description |
| --- | --- |
| TDE2_FILTER_MODE_BOTH | Filters the color and alpha channel concurrently. |
| TDE2_FILTER_MODE_BUTT | Invalid |

[Note]

The picture scaling or anti-flicker operation is a filtering operation. Therefore, you need to specify the filtering mode before performing the scaling or/and anti-flicker operations.

[See Also]

None

## TDE2_FILT_COEF_S

[Description]

Defines the filtering coefficient.

[Syntax]

```
typedef struct hiTDE2_FILT_COEF_S
{
    TDE2_FILT_COEF_TYPE_E enFilterType;
    TDE2_HFILT_COEF_E enHFilterL;
    TDE2_HFILT_COEF_E enHFilterC;
    TDE2_VFILT_COEF_E enVFilterL;
    TDE2_VFILT_COEF_E enVFilterC;
}TDE2_FILT_COEF_S;
```

[Member]

| Member | Description |
| --- | --- |
| enFilterType | Filtering coefficient type |
| enHFilterL | Horizontal filtering coefficient for luminance |
| enHFilterC | Horizontal filtering coefficient for chrominance |
| enVFilterL | Vertical filtering coefficient for luminance |
| enVFilterC | Vertical filtering coefficient for chrominance |

[Note]

None

[See Also]

None

## TDE2_FILT_COEF_TYPE_E

[Description]

Defines the filtering coefficient type.

[Syntax]

```
typedef enum hiTDE2_FILT_COEF_TYPE_E
{
    TDE2_FILT_COEF_TYPE_NORM = 0,
    TDE2_FILT_COEF_TYPE_EX,
    TDE2_FILT_COEF_TYPE_EX2,
    TDE2_FILT_COEF_TYPE_USER1,
    TDE2_FILT_COEF_TYPE_USER2,
    TDE2_FILT_COEF_TYPE_BUTT
}TDE2_FILT_COEF_TYPE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_FILT_COEF_TYPE_NORM | Universal filtering type. It is applicable to common filtering tasks. |
| TDE2_FILT_COEF_TYPE_EX | Extended filtering type. It is obtained by convoluting the VI filtering coefficient and norm coefficient. |
| TDE2_FILT_COEF_TYPE_EX2 | Extended filtering type 2 |
| TDE2_FILT_COEF_TYPE_USER1 | User-defined filtering coefficient type 1 |
| TDE2_FILT_COEF_TYPE_USER2 | User-defined filtering coefficient type 2 |
| TDE2_FILT_COEF_TYPE_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_FILLCOLOR_S

[Description]

Defines the attributes of the picture fill colors.

[Syntax]

```
typedef struct hiTDE2_FILLCOLOR_S
{
    TDE2_COLOR_FMT_E enColorFmt;
    HI_U32           u32FillColor;
} TDE2_FILLCOLOR_S;
```

[Member]

| Member | Description |
|---|---|
| enColorFmt | Format of the fill color |
| u32FillColor | Fill value |

[Note]

The fill value must match the format of the fill color. For example, if you want to fill blue in a bitmap, you can specify the format of the fill color to ARGB15555 and set the fill value to 0x801F (the alpha bit is 1).

[See Also]

None

## TDE2_MIRROR_E

[Description]

Defines the mirror attributes of a picture.

[Syntax]

```
typedef enum hiTDE2_MIRROR_E
{
    TDE2_MIRROR_NONE = 0,
    TDE2_MIRROR_HORIZONTAL,
    TDE2_MIRROR_VERTICAL,
    TDE2_MIRROR_BOTH,
    TDE2_MIRROR_BUTT
} TDE2_MIRROR_E
```

[Member]

| Member | Description |
|---|---|
| TDE2_MIRROR_NONE | Does not perform the mirror operation on the output picture. |
| TDE2_MIRROR_HORIZONTAL | Performs the horizontal mirror operation on the output picture. |

| Member | Description |
| --- | --- |
| TDE2_MIRROR_VERTICAL | Performs the vertical mirror operation on the output picture. |
| TDE2_MIRROR_BOTH | Performs the horizontal and vertical mirror operations on the output picture concurrently. |
| TDE2_MIRROR_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_SURFACE_S

[Description]

Defines the surface of a bitmap.

[Syntax]

```
typedef struct hiTDE2_SURFACE_S
{
    HI_U32 u32PhyAddr;
    TDE2_COLOR_FMT_E enColorFmt;
    HI_U32 u32Height;
    HI_U32 u32Width;
    HI_U32 u32Stride;
    HI_U8* pu8ClutPhyAddr;
    HI_BOOL bYCbCrClut;
    HI_BOOL bAlphaMax255;
    HI_BOOL bAlphaExt1555;
    HI_U8 u8Alpha0;
    HI_U8 u8Alpha1;
} TDE2_SURFACE_S;
```

[Member]

| Member | Description |
| --- | --- |
| u32PhyAddr | Start address of a bitmap. |
| enColorFmt | Bitmap format. |
| u16Height | Bitmap height. |
| u16Width | Bitmap width. |

| Member | Description |
| --- | --- |
| u16Stride | Bitmap stride. |
| pu8ClutPhyAddr | Start address of the CLUT, for color extension or color correction. |
| bYCbCrClut | Whether the CLUT is in the YCbCr space. |
| bAlphaMax255 | The maximum alpha value of a bitmap is 255 or 128.<br>Note that the maximum alpha value is 255 for the Hi3518. |
| bAlphaExt1555 | Whether to enable the alpha extension of an ARGB1555 bitmap.<br>The parameter is valid if the bitmap is in the ARGB1555 format. |
| u8Alpha0 | Alpha0 value.<br>Value range: [0, 255].<br>The parameter is valid if the bitmap is in the ARGB1555 format and bAlphaExt1555 is set to TRUE.<br>When the format is ARGB1555, if the most significant bit (MSB) of the pixel is 0, the alpha0 value is selected as the alpha value for alpha blending. |
| u8Alpha1 | Alpha1 value.<br>Value range: [0, 255].<br>The parameter is valid if the bitmap is in the ARGB1555 format and bAlphaExt1555 is set to TRUE.<br>When the format is ARGB1555, if the MSB of the pixel is 1, the alpha1 value is selected as the alpha value for alpha blending. |

[Note]

- If the pixel format of a bitmap is greater than or equal to a byte, the start address and stride of the bitmap format must be aligned based on the pixel format. If the pixel format of a bitmap is smaller than a byte, the start address and stride of the bitmap must be aligned based on byte.
- If the pixel format of a bitmap is smaller than a byte, the horizontal start point and width of the bitmap must be aligned based on pixel.
- The horizontal start point and width of a YCbCr422 bitmap must be even numbers.
- The extension from the CLUT to the true color ARGB is implemented by retrieving the CLUT. Therefore, for the color extension function (for example, extend a CLUT1 bitmap to an ARGB8888 bitmap) or the color correction function, you need to configure the start address pu8ClutPhyAddr of the CLUT and ensure that the memory corresponding to the start address is physically continuous.

[See Also]

None

## TDE2_OPT_S

[Description]

Defines the attributes of a TDE operation.

[Syntax]

```
typedef struct hiTDE2_OPT_S
{
    TDE2_ALUCMD_E enAluCmd;                 /*Logical operation type*/
    TDE2_ROP_CODE_E enRopCode_Color;        /*ROP type of the color space*/
    TDE2_ROP_CODE_E enRopCode_Alpha;        /*ROP type of the alpha*/
    TDE2_COLORKEY_MODE_E enColorKeyMode;    /*Colorkey mode*/
    TDE2_COLORKEY_U unColorKeyValue;        /*Colorkey value*/
    TDE2_CLIPMODE_E enClipMode;             /*Intra-area clip or extra-area
clip*/
    TDE2_RECT_S stClipRect;                 /*Definition of the clipped
area*/
    HI_BOOL bDeflicker;                     /*Whether to perform
anti-flicker*/
    TDE2_DEFLICKER_MODE_E enDeflickerMode;  /**<Anti-flicker mode*/
    TDE2_FILTER_MODE_E enFilterMode;        /*Filtering mode for scaling or
anti-flicker*/
    TDE2_MIRROR_E enMirror;                 /*Mirror type*/
    HI_BOOL bClutReload;                    /*Whether to reload the CLUT*/
    HI_U8   u8GlobalAlpha;                  /*Global alpha value*/
    TDE2_OUTALPHA_FROM_E enOutAlphaFrom;    /*Alpha output source*/
    HI_U32 u32Colorize;                  /**<Colorize value*/
    TDE2_BLEND_OPT_S stBlendOpt;
    TDE2_CSC_OPT_S stCscOpt;
} TDE2_OPT_S
```

[Member]

| Member | Description |
|---|---|
| enAluCmd | Logical operation type |
| enRopCode_Color | ROP type of the color space |
| enRopCode_Alpha | ROP type of the alpha |
| enColorKeyMode | Colorkey mode |
| unColorKeyValue | Colorkey value |
| enClipMode | Intra-area clip or extra-area clip |
| stClipRect | Definition of the clipped area |
| bDeflicker | Whether to perform anti-flicker |
| bResize | Whether to scale |
| enFilterMode | Filtering mode for scaling or anti-flicker |

| Member | Description |
| --- | --- |
| enMirror | Mirror type |
| bClutReload | Whether to reload the CLUT |
| u8GlobalAlpha | Global alpha value<br>Value range: [0, 255] |
| enOutAlphaFrom | Alpha output source |
| u32Colorize | Colorize value |
| stBlendOpt | Alpha blending operation option |
| stCscOpt | CSC parameter option |

[Note]

None

[See Also]

None

## TDE2_MB_COLOR_FMT_E

[Description]

Defines the macroblock format supported by the TDE.

[Syntax]

```
typedef enum hiTDE2_MB_COLOR_FMT_E
{
    TDE2_MB_COLOR_FMT_JPG_YCbCr400MBP = 0,
    TDE2_MB_COLOR_FMT_JPG_YCbCr422MBHP,
    TDE2_MB_COLOR_FMT_JPG_YCbCr422MBVP,
    TDE2_MB_COLOR_FMT_MP1_YCbCr420MBP,
    TDE2_MB_COLOR_FMT_MP2_YCbCr420MBP,
    TDE2_MB_COLOR_FMT_MP2_YCbCr420MBI,
    TDE2_MB_COLOR_FMT_JPG_YCbCr420MBP,
    TDE2_MB_COLOR_FMT_JPG_YCbCr444MBP,
    TDE2_MB_COLOR_FMT_BUTT
} TDE2_MB_COLOR_FMT_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE2_MB_COLOR_FMT_JPG_YCbCr400MBP | Macroblock 400 in the JPEG encoding format |

| Member | Description |
| --- | --- |
| TDE2_MB_COLOR_FMT_JPG_YCbCr422MBHP | Macroblock 422 in the JPEG encoding format (half of the horizontal sampling) |
| TDE2_MB_COLOR_FMT_JPG_YCbCr422MBVP | Macroblock 422 in the JPEG encoding format (half of the vertical sampling) |
| TDE2_MB_COLOR_FMT_MP1_YCbCr420MBP | Macroblock 420 in the MPEG-1 encoding format |
| TDE2_MB_COLOR_FMT_MP2_YCbCr420MBP | Macroblock 420 in the MPEG-2 encoding format |
| TDE2_MB_COLOR_FMT_MP2_YCbCr420MBI | Macroblock 420 in the MPEG-2 encoding format (interlaced) |
| TDE2_MB_COLOR_FMT_JPG_YCbCr420MBP | Macroblock 420 in the JPEG encoding format |
| TDE2_MB_COLOR_FMT_JPG_YCbCr444MBP | Macroblock 444 in the JPEG encoding format |

[Note]

None

[See Also]

None

## TDE_PIC_MODE_E

[Description]

Defines the processing mode of the frame or field supported by the TDE.

[Syntax]

```
typedef enum hiTDE_PIC_MODE_E
{
    TDE_FRAME_PIC_MODE = 0,     /*Processing mode of the frame*/
    TDE_BOTTOM_FIELD_PIC_MODE, /*Processing mode of the bottom field*/
    TDE_TOP_FIELD_PIC_MODE,     /*Processing mode of the top field*/
    TDE_TB_FIELD_PIC_MODE,      /*Processing mode of the top and bottom
fields*/
    TDE_PIC_MODE_BUTT
} TDE_PIC_MODE_E;
```

[Member]

| Member | Description |
| --- | --- |
| TDE_FRAME_PIC_MODE | Processing mode of the frame |

| Member | Description |
|---|---|
| TDE_BOTTOM_FIELD_PIC_MODE | Processing mode of the bottom field |
| TDE_TOP_FIELD_PIC_MODE | Processing mode of the top field |
| TDE_TB_FIELD_PIC_MODE | Processing mode of the top and bottom fields |
| TDE_PIC_MODE_BUTT | Invalid |

[Note]

None

[See Also]

None

## TDE2_MB_S

[Description]

Defines the surface of the macroblock. This data type describes the basic information about the picture in macroblock format.

[Syntax]

```
typedef struct hiTDE2_MB_S
{
    TDE2_MB_COLOR_FMT_E enMbFmt;
    HI_U32              u32YPhyAddr;
    HI_U32              u32YWidth;
    HI_U32              u32YHeight;
    HI_U32              u32YStride;
    HI_U32              u32CbCrPhyAddr;
    HI_U32              u32CbCrStride;
    TDE2_COLORSPACE_CONV_MODE_E enColorSpaceConv;  /*Matrix for color space conversion*/
    TDE_PIC_MODE_E      enPicMode;
} TDE2_MB_S;
```

[Member]

| Member | Description |
|---|---|
| enMbFmt | Format of a macroblock |
| u32YPhyAddr | Start physical address of a luminance block |
| u32YWidth | Width of a luminance block |
| u32YHeight | Height of a luminance block |

| Member | Description |
|---|---|
| u32YStride | Stride between adjacent lines of a luminance block |
| u32CbCrPhyAddr | Start physical address of a chrominance block |
| u32CbCrStride | Stride between adjacent lines of a chrominance block |
| enColorSpaceConv | Matrix for color space conversion |
| enPicMode | Processing mode of the frame or field of a bitmap |

[Note]

None

[See Also]

None

## TDE2_COLORSPACE_CONV_MODE_E

[Description]

Defines the mode of the color space conversion from the macroblock format to the raster format.

[Syntax]

```
typedef enum hiTDE2_COLORSPACE_CONV_MODE_E
{
    TDE2_ITU_R_BT601_IMAGE = 0,
    TDE2_ITU_R_BT709_IMAGE,
    TDE2_ITU_R_BT601_VIDEO,
    TDE2_ITU_R_BT709_VIDEO
}TDE2_COLORSPACE_CONV_MODE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_ITU_R_BT601_IMAGE | BT601 picture conversion standard during conversion |
| TDE2_ITU_R_BT709_IMAGE | BT709 picture conversion standard during conversion |
| TDE2_ITU_R_BT601_VIDEO | BT601 video conversion standard during conversion |
| TDE2_ITU_R_BT709_VIDEO | BT709 video conversion standard during conversion |

[Note]

None

[See Also]

None

## TDE2_MBRESIZE_E

[Description]

Defines the scaling mode of the macroblock format.

[Syntax]

```
typedef enum hiTDE2_MBRESIZE_E
{
    TDE2_MBRESIZE_NONE = 0,
    TDE2_MBRESIZE_QUALITY_LOW,
    TDE2_MBRESIZE_QUALITY_MIDDLE,
    TDE2_MBRESIZE_QUALITY_HIGH,
    TDE2_MBRESIZE_BUTT
} TDE2_MBRESIZE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_MBRESIZE_NONE | No scaling |
| TDE2_MBRESIZE_QUALITY_LOW | Low-quality scaling mode of the macroblock surface |
| TDE2_MBRESIZE_QUALITY_MIDDLE | Medium-quality scaling mode of the macroblock surface |
| TDE2_MBRESIZE_QUALITY_HIGH | High-quality scaling mode of the macroblock surface |

[Note]

None

[See Also]

None

## TDE2_MBOPT_S

[Description]

Defines the attributes of the macroblock surface operation.

[Syntax]

```
typedef struct hiTDE2_MBOPT_S
{
    TDE2_CLIPMODE_E enClipMode;
```

```
    TDE2_RECT_SstClipRect;
    HI_BOOL bDeflicker;
    TDE2_MBRESIZE_E enResize;
    HI_BOOL bSetOutAlpha;
    HI_U8   u8OutAlpha;
    TDE2_FILT_COEF_S stFiltCoef;
    HI_BOOL bForceHFilt;
    HI_BOOL bForceVFilt;
} TDE2_MBOPT_S;
```

[Member]

| Member | Description |
|---|---|
| enClipMode | Clip mode: intra-area clip or extra-area clip. |
| stClipRect | Definition of the clipped area. |
| bDeflicker | Whether to perform anti-flicker. |
| enResize | Macroblock scaling mode: no scaling, high-quality scaling, medium-quality scaling, or high-quality scaling. |
| bSetOutAlpha | Whether the alpha value of the output result bitmap is specified by users.<br>If the alpha value is not set, the maximum alpha value is output by default. |
| u8OutAlpha | Alpha value of the output result bitmap specified by users. |
| stFiltCoef | Filtering coefficient select. |
| bForceHFilt | Whether to forcibly perform horizontal filtering during 1:1 transfer. |
| bForceVFilt | Whether to forcibly perform vertical filtering during 1:1 transfer. |

[Note]

None

[See Also]

None

## TDE2_MBFILL_E

[Description]

Defines the attributes of the macroblock surface operation.

[Syntax]

```
typedef enum
{
```

```
    TDE2_MBFILL_YC = 0, /*Fill in chrominance and luminance blocks*/
    TDE2_MBFILL_Y,      /*Fill in only the luminance block*/
    TDE2_MBFILL_C,      /*Fill in only the chrominance block*/
}TDE2_MBFILL_E;
```

[Member]

| Member | Description |
|---|---|
| TDE2_MBFILL_YC | Fill in chrominance and luminance blocks. |
| TDE2_MBFILL_Y | Fill in only the luminance block. |
| TDE2_MBFILL_C | Fill in only the chrominance block. |

[Note]

None

[See Also]

None

## TDE2_CSC_OPT_S

[Description]

Defines CSC parameter options.

[Syntax]

```
typedef struct hiTDE2_CSC_OPT_S
{
    HI_BOOL bICSCUserEnable;        /**User-defined ICSC parameter enable*/
    HI_BOOL bICSCParamReload;   /**User-defined ICSC parameter reload
enable*/
    HI_BOOL bOCSCUserEnable;        /**User-defined OCSC parameter enable*/
    HI_BOOL bOCSCParamReload;   /**User-defined OCSC parameter reload
enable*/
    HI_U32 u32ICSCParamAddr;        /**ICSC parameter address. The address
must be 128-bit aligned.*/
    HI_U32 u32OCSCParamAddr;    /**OCSC parameter address. The address must
be 128-bit aligned.*/
}TDE2_CSC_OPT_S;
```

[Member]

| Member | Description |
|---|---|
| bICSCUserEnable | User-defined ICSC parameter enable. |
| bICSCParamReload | User-defined ICSC parameter reload enable. |

| Member | Description |
|---|---|
| bOCSCUserEnabl | User-defined OCSC parameter enable. |
| bOCSCParamReload | User-defined OCSC parameter reload enable. |
| u32ICSCParamAddr | ICSC parameter address. The address must be 128-bit aligned. |
| u32OCSCParamAddr | OCSC parameter address. The address must be 128-bit aligned. |

[Note]

None

[See Also]

None

## TDE2_ROTATE_ANGLE_E

[Description]

Defines the rotation angle.

[Syntax]

```
typedef enum hiTDE_ROTATE_ANGLE_E
{
    TDE_ROTATE_CLOCKWISE_90 = 0x0, /*Rotation by 90°*/
    TDE_ROTATE_CLOCKWISE_180,     /*Rotation by 180°*/
    TDE_ROTATE_CLOCKWISE_270,     /*Rotation by 270°*/
    TDE_ROTATE_BUTT
}TDE_ROTATE_ANGLE_E;
```

[Member]

| Member | Description |
|---|---|
| TDE_ROTATE_CLOCKWISE_90 | Clockwise rotation by 90º. |
| TDE_ROTATE_CLOCKWISE_180 | Clockwise rotation by 180º. |
| TDE_ROTATE_CLOCKWISE_270 | Clockwise rotation by 270º. |
| TDE_ROTATE_BUTT | Invalid. |

[Note]

None

[See Also]

None

## LDC_ATTR_S

[Description]

Defines the distortion correction attribute.

For details, see chapter 3 "VI" in the *HiMPP Media Processing Software Development Reference*.

# 4 Error Codes

Table 4-1 describes the error codes of TDE APIs.

**Table 4-1** Error codes of TDE APIs

| Error Code | Macro Definition | Description |
|---|---|---|
| 0xA0648001. | HI_ERR_TDE_DEV_NOT_OPEN | The TDE device is not started. |
| 0xA0648002 | HI_ERR_TDE_DEV_OPEN_FAILED | The TDE device fails to be started. |
| 0xA0648003 | HI_ERR_TDE_NULL_PTR | The pointer of the input parameter is null. |
| 0xA0648004 | HI_ERR_TDE_NO_MEM | The memory fails to be allocated. |
| 0xA0648005 | HI_ERR_TDE_INVALID_HANDLE | The task handle is invalid. |
| 0xA0648006 | HI_ERR_TDE_INVALID_PARA | The input parameter is invalid. |
| 0xA0648007 | HI_ERR_TDE_NOT_ALIGNED | The position, width, height, or stride of the picture is not aligned as required. |
| 0xA0648008 | HI_ERR_TDE_MINIFICATION | The multiple of down scaling exceeds the limitation (the maximum value is 255). |
| 0xA0648009 | HI_ERR_TDE_CLIP_AREA | The operation area does not overlap the clipped area. |
| 0xA064800A | HI_ERR_TDE_JOB_TIMEOUT | The waiting times out. |
| 0xA064800B | HI_ERR_TDE_UNSUPPORTED_OPERATION | The operation is not supported. |
| 0xA064800C | HI_ERR_TDE_QUERY_TIMEOUT | The specific task is not complete due to timeout. |
| 0xA064800D | HI_ERR_TDE_EMPTY_JOB | The task is empty. |

# 5 Instances

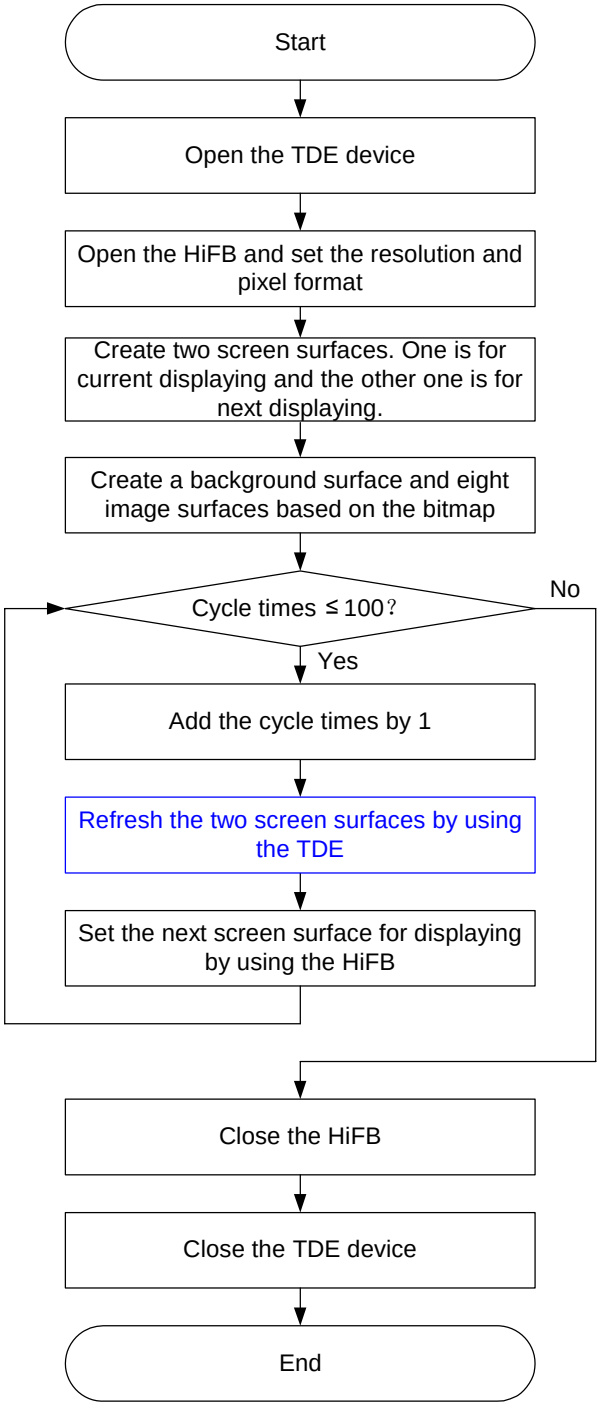
## 5.1 Software Process

**NOTE**

This section describes how to implement picture animation rotation by using the bitblt and color space. Ensure that the TDE and HiSilicon frame buffer (HiFB) drivers are loaded and the video output device works properly before enabling the TDE. In this instance, you need to allocate at least 1658880 bytes for the display buffer of graphics layer 0. For details about how to load the HiFB, see the *HiFB Development Guide*.

Figure 5-1 shows the software process.

**Figure 5-1** Software process (main process)

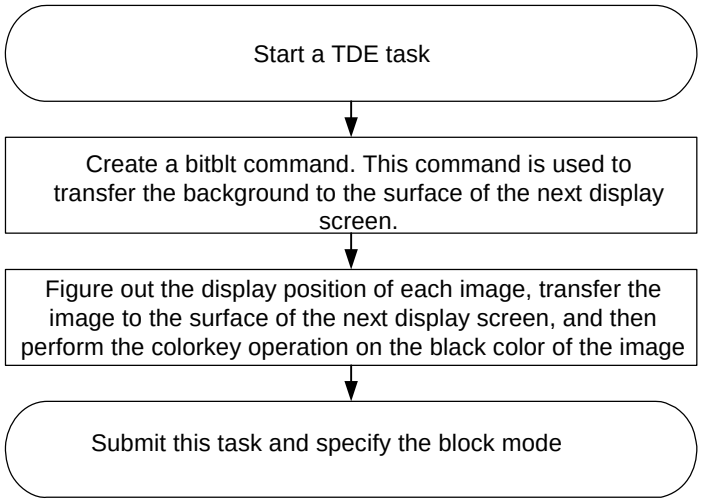



**NOTE**

For details about how to refresh the two screen surfaces by using the TDE, see Figure 5-2.

Figure 5-2 shows how to refresh the two screen surfaces by using the TDE.

**Figure 5-2** Refreshing the two screen surfaces by using the TDE




# 5.2 Reference Codes

For details about the codes, see the **tde/sample_tde.c** in the **sample** folder of the SDK.