Hi3520 TDE

API 参考

文档版本	00B10
发布日期	2009-10-30
BOM编码	N/A

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为电气生产中心　邮编：518129

网址：　http://www.hisilicon.com

客户服务电话：　+86-755-28788858

客户服务传真：　+86-755-28357515

客户服务邮箱：　support@hisilicon.com

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

插图目录

前 言

概述

本文为利用 Hi3520 媒体处理芯片进行图形开发的程序员而写，介绍了二维图像加速模块（TDE）的重要概念、API 使用方法、数据结构及实例等。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3520 H.264 编解码处理器	V100

读者对象

本文档主要适用于以下工程师：

- 技术支持工程师
- 图形软件开发工程师

约定

通用格式约定

格式	说明
宋体	正文采用宋体表示。
黑体	一级、二级、三级标题采用黑体。
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。
“Terminal Display”格式	“Terminal Display”格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。

命令行格式约定

格式	意义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用加粗字体表示。
斜体	命令行参数（命令中必须由实际值进行替代的部分）采用*斜体*表示。
[ ]	表示用“[ ]”括起来的部分在命令配置时是可选的。
{ x | y | ... }	表示从两个或多个选项中选取一个。
[x | y | ...]	表示从两个或多个选项中选取一个或者不选。
{ x | y | ... } *	表示从两个或多个选项中选取多个，最少选取一个，最多选取所有选项。
[x | y | ...] *	表示从两个或多个选项中选取多个或者不选。

表格内容约定

内容	说明
-	表格中的无内容单元。
*	表格中的内容用户可根据需要进行配置。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2009-10-30	00B10	基线版本。优化部分描述。
2009-09-30	00B02	补充部分 API 和数据结构；优化部分描述。
2009-08-29	00B01	第一次发布。

1 概述

1.1 概述

TDE（Two Dimensional Engine）利用硬件为 OSD（On Screen Display）和 GUI（Graphics User Interface）提供快速的图形绘制功能，主要有快速位图搬移、快速色彩填充、快速抗闪搬移、快速位图缩放、画点、画水平/垂直线、位图格式转换、位图 alpha 叠加、位图按位布尔运算、ColorKey 操作。

1.2 参考域说明

1.2.1 API 参考域

本手册使用 9 个参考域描述 API 的相关信息，它们的作用如表 1-1 所示。

表1-1 API 参考域说明

参考域	含义
目的	简要描述 API 的主要功能。
语法	列出调用 API 应包括的头文件以及 API 的原型声明。
参数	列出 API 的参数、参数说明及参数属性。
描述	简要描述 API 的工作过程。
返回值	列出 API 所有可能的返回值及其含义。
需求	列出 API 包含的头文件和 API 编译时要链接的库文件。
注意	列出使用 API 时应注意的事项。
举例	列出使用 API 的实例。
相关接口	列出与本 API 相关联的其他接口。

1.2.2 数据类型参考域

本手册使用 5 个参考域描述数据类型的相关信息，它们的作用如表 1-2 所示。

表1-2 数据类型参考域说明

参考域	含义
说明	简要描述数据类型的主要功能。
定义	列出数据类型的定义语句。
成员	列出数据结构的成员及含义。
注意事项	列出使用数据类型时应注意的事项。
相关数据类型和接口	列出与本数据类型相关联的其他数据类型和接口。

2 API 参考

2.1 API 概述

TDE（Two Dimension Engine）功能函数参考提供 2D 加速相关操作。

该功能模块提供以下 API：

- HI_TDE2_Open：打开 TDE 设备。
- HI_TDE2_Close：关闭 TDE 设备。
- HI_TDE2_BeginJob：创建 1 个 TDE 任务。
- HI_TDE2_EndJob：提交添加操作完成的 TDE 任务。
- HI_TDE2_QuickCopy：向任务中添加快速拷贝操作。
- HI_TDE2_QuickFill：向任务中添加快速填充操作。
- HI_TDE2_QuickResize：向任务中添加光栅位图缩放操作。
- HI_TDE2_QuickDeflicker：向任务中添加抗闪烁操作。
- HI_TDE2_Bitblit：向任务中添加对光栅位图进行有附加功能的搬移操作。
- HI_TDE2_MbBlit：向任务中添加对宏块位图进行有附加功能的搬移操作。
- HI_TDE2_SolidDraw：向任务中添加对光栅位图进行有附加操作的填充搬移操作。
- HI_TDE2_BitmapMaskRop：向任务中添加对光栅位图进行 Mask Rop 搬移操作。
- HI_TDE_BitmapMaskBlend：向任务中添加对光栅位图进行 Mask Blend 搬移操作。
- HI_TDE2_CancelJob：取消指定的 TDE 任务。
- HI_TDE2_WaitForDone：等待指定的 TDE 任务完成。

2.2 功能函数参考

HI_TDE2_Open

【目的】

打开 TDE 设备。

【语法】

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

【描述】

调用此接口打开 TDE 设备。

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_TDE_DEV_OPEN_FAILED	TDE 设备打开失败。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在进行 TDE 相关操作前应该首先调用此接口，保证 TDE 设备处于打开状态。
- TDE 设备允许多进程重复打开。

【举例】

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

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

```
/* close TDE device*/
HI_TDE2_Close();
```

HI_TDE2_Close

【目的】

关闭 TDE 设备。

【语法】

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

【描述】

调用此接口关闭 TDE 设备。

【参数】

无。

【返回值】

无。

【错误码】

无。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

调用 HI_TDE2_Open 与 HI_TDE2_Close 的次数需要对应。

【举例】

无。

HI_TDE2_BeginJob

【目的】

创建 1 个 TDE 任务。

【语法】

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

【描述】

调用此接口创建 1 个 TDE 任务（Job）。TDE 以任务的形式管理 TDE 命令：1 个 TDE 任务是一系列 TDE 命令的集合，它可以包含 1 个或多个 TDE 操作；一个 TDE 命令对

应一个 TDE 操作；成功创建 TDE 任务添加完 TDE 操作后，通过 HI_TDE2_EndJob 提交该 Job；同一任务中的 TDE 命令是顺序执行。

【参数】

无。

【返回值】

返回值	描述
TDE_HANDLE	合法的 TDE 任务句柄
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备没有打开
HI_ERR_TDE_INVALID_HANDLE	无效的 TDE 任务句柄

【错误码】

接口返回值	含义
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_INVALID_HANDLE	非法的 TDE 任务句柄。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应确保 TDE 设备处于打开状态。
- 应判断返回值，确保获得 1 个正确的任务句柄。
- TDE 能够缓存的任务数最多为 200 个。
- HI_TDE2_BeginJob 必须和 HI_TDE2_EndJob 配套使用，否则会造成内存泄漏。

【举例】

```
/* 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_EndJob

【目的】

提交已创建的 TDE 任务。

【语法】

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

【描述】

调用此接口提交 1 个 TDE 任务。可以指定为阻塞还是非阻塞，阻塞时可以设置超时。

- 阻塞

  指该函数调用不会立刻返回，只有在以下情况下才会返回：

  - TDE 任务中的命令都完成
  - 等待超时
  - 等待被中断
- 非阻塞

  指该函数调用会立刻返回，而不关心 TDE 任务中的命令是否已经完成。

阻塞时可以设置一个最长等待时间，如果等待时间到了，TDE 任务中的命令还没有完成，函数就会提前返回，但是任务中的命令还是会在未来的某个时刻完成。

【参数】

说明

Hi3520 芯片不支持同步提交方式，故参数 bSync 需配置为 HI_FALSE。

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
bSync	是否使用同步方式提交。 HI_TRUE：同步方式。 HI_FALSE：非同步方式。	输入	局部

参数名称	描述	输入/输出	全局/局部
bBlock	阻塞标志。 HI_TRUE：阻塞。 HI_FALSE：非阻塞。	输入	局部
u32TimeOut	超时时间，单位 jiffies（10ms）。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_SUCCESS	任务提交成功。 • 阻塞任务：任务中的所有 TDE 命令已经完成。 • 非阻塞任务：任务中的所有 TDE 命令都已经提交成功。
HI_ERR_TDE_INVALID_HANDLE	输入的 Job handler 无效。
HI_ERR_TDE_JOB_TIMEOUT	等待超时。
HI_ERR_TDE_EMPTY_JOB	提交空操作的任务。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 若设置为阻塞操作，函数超时返回或被中断返回时应该注意：此时 TDE 操作的 API 函数提前返回，但执行的操作仍会完成。
- 提交任务后，此任务对应的 handle 会变为无效，再次提交会出现错误码 HI_ERR_TDE_INVALID_HANDLE。

- 不允许提交一个不包含任何操作的任务，否则返回错误码 HI_ERR_TDE_EMPTY_JOB。

【举例】

无。

HI_TDE2_QuickCopy

【目的】

向指定任务中添加快速拷贝操作。

【语法】

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

【描述】

将基地址为 pstSrc 的位图的指定区域 pstSrcRect 拷贝到以 pstDst 为目的地址、pstDstRect 为输出区域的内存中。

位图、位图中的操作区域及两者的的关系描述如下：

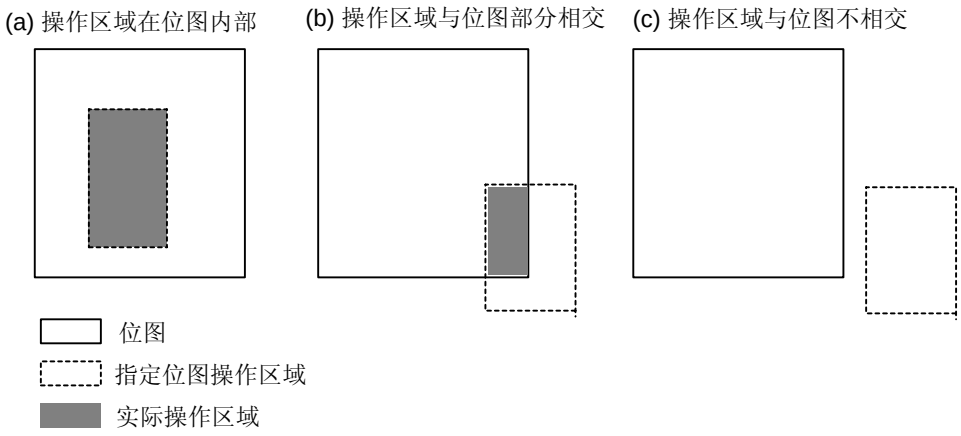
- 位图信息由 TDE2_SURFACE_S 表示，它描述位图的基本信息，包括：位图的像素宽度、像素高度、位图每行的跨度、颜色格式、位图存放的物理地址等。
- 操作区域由 TDE2_RECT_S 表示，它描述位图中参与本次操作的矩形范围，包括：起始位置和尺寸信息。
- 位图及位图操作区域的关系如图 2-1 所示。

  通过指定不同的操作区域，用户可指定位图的全部或部分参与操作。
  - 若希望整个位图都参与操作，则指定操作区域起点为（0，0），宽高分别为位图的宽高。
  - 若希望位图的部分区域参与操作，则指定适当的操作区域大小。如图 2-1 中的情况（a），指定的操作区域即为有效的操作区域。注意：如果指定的操作区域与位图部分相交，如情况(b)，则自动裁剪指定操作区域，故有效的操作区域为灰色相交部分。
  - 若指定的操作区域与位图不相交，如情况(c)，则认为配置错误，返回错误码 HI_ERR_TDE_INVALID_PARA。

说明

有效操作区域：指调用者指定的操作区域与位图的相交部分。

图2-1 位图与位图中的操作区域的关系



【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstSrc	源位图。	输入	局部
pstSrcRect	源位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。

返回值	描述
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 此接口使用的是直接 DMA 搬移，因此性能优于 HI_TDE2_Bitblit 搬移。
- 快速拷贝操作不支持格式转换，源位图和目标位图格式必须一致。
- 快速拷贝不支持缩放功能，因此如果源和目的的操作区域尺寸不一致，则按照两者最小的公共区域进行拷贝搬移。
- 指定的操作区域要和指定的位图有公共区域，否则会返回错误；其他操作均有此要求。
- 像素格式大于等于 Byte 的位图格式的基地址和位图的 Stride 必须按照像素格式对齐，像素格式不足 Byte 的位图格式的基地址和 Stride 需要按照 Byte 对齐；其他操作均有此要求。
- 像素格式不足 Byte 的位图格式的水平起始位置和宽度必须按照像素对齐。
- YCbCr422 格式的位图的水平起始位置和宽度必须为偶数；其他操作均有此要求。

【举例】

无。

HI_TDE2_QuickFill

【目的】

向任务中添加快速填充操作。

【语法】

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

【描述】

将数据值 u32FillData 填充到以 pstDst 为目的地址、pstDstRect 为输出区域的内存中，可实现颜色填充的功能。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部
u32FillData	填充值。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

由于该操作直接将 u32FillData 填充在位图的指定区域内，调用者欲填充蓝色到指定位图，应按照位图格式指定相应的蓝色填充值。

如位图格式为 ARGB1555，欲填充为蓝色，则应指定 u32FillData 为 0x801F（其中 alpha 位为 1）。

【举例】

无。

HI_TDE2_QuickResize

【目的】

向任务中添加光栅位图缩放操作。

【语法】

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

【描述】

将基地址为 pstSrc 的位图以区域 pstSrcRect 指定的尺寸缩放至 pstDstRect 的尺寸，将结果拷贝到以 pstDst 为目的地址、pstDstRect 为输出区域的内存中。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstSrc	源位图。	输入	局部
pstSrcRect	源位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。

返回值	描述
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_ERR_TDE_MINIFICATION	缩小倍数过大。
HI_ERR_TDE_NOT_ALIGNED	Clut 表的起始地址没有按照 4byte 对齐。
HI_ERR_TDE_UNSUPPORTED_OPERATION	不支持的操作配置。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_stb_api.h
- 库文件：libhimpi.a

【注意】

- 缩小倍数最大为 15 倍，放大倍数则没有限制。
- 缩放时源位图和目标位图可以为同一位图，但不能为内存有重叠的不同位图。
- 如果源位图和目标位图的格式不相同，则自动进行格式转换。

【举例】

无。

HI_TDE2_QuickDeflicker

【目的】

向任务中添加抗闪烁操作。

【语法】

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

【描述】

将基地址为 pstSrc 的位图以指定的区域 pstSrcRect 进行抗闪烁，将结果拷贝到以 pstDst 为目的地址、pstDstRect 为输出区域的内存中。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstSrc	源位图。	输入	局部
pstSrcRect	源位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_ERR_TDE_NOT_ALIGNED	位图起始地址或 Stride 没有按照像素对齐。
HI_ERR_TDE_UNSUPPORTED_OPERATION	不支持的操作配置。
HI_ERR_TDE_MINIFICATION	缩小倍数过大。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 抗闪烁只按垂直方向进行滤波。
- 抗闪烁的源和目的位图可以在同一块内存。
- 如果指定的输入区域与输出不一致，则会进行缩放处理。
- 如果源位图和目标位图的格式不相同，则会进行格式转换处理。

【举例】

无。

HI_TDE2_Bitblit

【目的】

向任务中添加对光栅位图进行有附加功能的搬移操作。

【语法】

```
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);
```

【描述】

将前景位图（pstForeGround）与背景位图（pstBackGround）的指定区域（pstForeGroundRect、pstBackGroundRect）进行运算，将运算后的位图拷贝到目标位图（pstDst）的指定区域（pstDstRect）中。

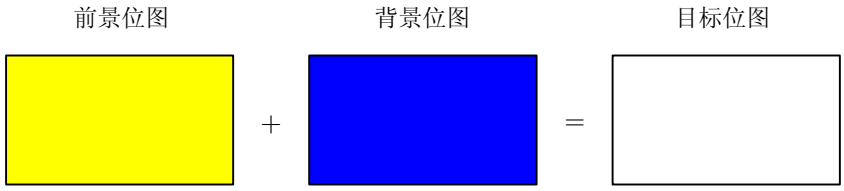
TDE2_OPT_S 结构中存放有 TDE 运算功能的配置信息，如：是否进行 ROP 操作及 ROP 命令码；是否作色键（colorkey）及 colorkey 的配置值；是否作区域裁减（clip 操作）及指定 clip 区域；是否缩放、是否抗闪烁、是否镜像、是否进行 alpha 混合等信息。

TDE2_OPT_S 结构中的配置项涉及到的概念解释如下：

- 按位布尔运算，即 ROP

  ROP 操作是指在将前景位图的 RGB 颜色分量和 alpha 分量值与背景位图的 RGB 颜色分量值和 alpha 分量值进行按位的布尔运算（包括按位与，按位或等），将结果输出，如图 2-2 所示。

图2-2 ROP 运算的搬移操作示意图（src1：R,G,B=0xFF,0xFF,0；src2：R,G,B=0,0,0xFF）



+：表示ROP或操作
=：表示运算后结果输出

- Alpha 混合操作

  Alpha 混合操作是指将前景位图和背景位图的像素值按照前景位图的 alpha 值进行加权求和，得到 1 个 alpha 混和后的输出位图，达到两个位图按照一定透明度叠加的效果。这里有 2 种方式叠加方式：

  说明
  无论哪种模式，全局 Alpha 都要参与叠加。

  - 前景位图数据是 Alpha 预乘后的数据，这时需要选择前景预乘 Alpha 叠加模式。

    Hi3520 不支持该种方式。

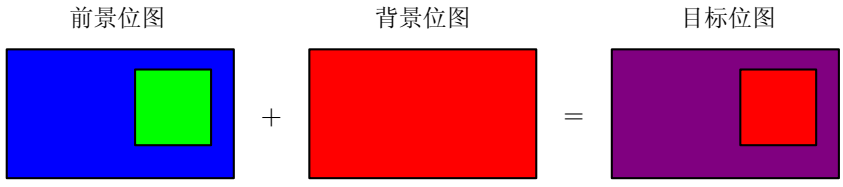
  - 前景位图数据没有预乘，这时需要选择前景非预乘 Alpha 叠加模式。

- ColorKey 操作

  Colorkey 操作是指满足关键色范围的像素不参与 TDE 运算。Colorkey 设置需要根据像素格式对每个分量单独设置过滤条件。所有分量满足过滤条件的颜色称为关键色。Colorkey 操作有 2 种模式：

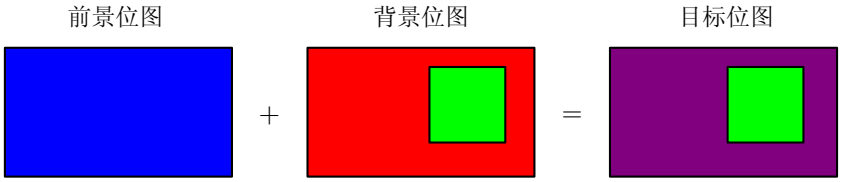
  - 对前景进行 colorkey，其设置的含义是前景位图中的关键色不参与运算，将背景位图保留，即背景位图相应区域直接拷贝至输出位图，如图 2-3 所示。
  - 对背景进行 colorkey，其设置的含义是背景位图中的关键色区域直接拷贝至输出位图，其他区域是运算结果，如图 2-4 所示。

图2-3 对前景位图进行 colorkey 运算的搬移操作示意图



+：表示Alpha运算并且对前景位图进行Colorkey操作
=：表示运算后结果输出

图2-4 对背景位图进行 colorkey 运算的搬移操作示意图



＋：表示Alpha运算并且对背景位图进行Colorkey操作
＝：表示运算后结果输出

- 缩放操作

  当前景位图中的操作区域若和目的位图的操作区域大小不一致时，有以下两种情况：

  - 若指定 TDE2_OPT_S 中 bResize 为 TRUE，即将前景的操作区域缩放到目的位图的操作区域大小，再和背景做其他运算。
  - 若指定 TDE2_OPT_S 中 bResize 为 FALSE，则不会进行前景操作区域的缩放，而是将前景、背景和目的位图的操作区域（pstForeGroundRect 、pstBackGroundRect、pstDstRect）三者求最小，并将该最小区域作为三个位图的真正操作区域。

- 抗闪烁操作

  抗闪烁操作指是否对前景位图的操作区域做抗闪处理，然后再和背景 alpha 叠加等运算，可通过 TDE2_OPT_S 中 bDeflicker 指定。

- 镜像功能

  镜像功能指将输出结果做水平（或/和垂直）方向上的反向。调用者可通过 TDE2_OPT_S 中 enMirror 配置项指定。镜像可以分为：

  - 水平镜像，按照水平方向进行对称拷贝。
  - 垂直镜像，按照垂直方向进行对称拷贝。
  - 水平垂直同时镜像，水平垂直同时对称拷贝。

- 颜色扩展或校正功能

  颜色扩展功能指将精度低的色彩格式通过调色板（称为 CLUT 表）扩展到真彩色。如 CLUT8 格式的位图仅有 256 色，调用者可通过构造合适的 CLUT 表，将 CLUT 表首地址配置给位图 SURFACE 的 pu8ClutPhyAddr 属性，TDE 就可以依靠检索 CLUT 表来实现 CLUT8 到真彩色 ARGB 的扩展。

  为了实现颜色扩展，调用者需要配置的选项为：

  - 位图 SURFACE 结构中的 CLUT 表首地址 pu8ClutPhyAddr，该地址指向的内存必须物理连续。
  - 位图 SURFACE 结构中的 bYCbCrClut 项，设置该 CLUT 表在 RGB 空间还是 YC 空间。
  - 操作结构 TDE2_OPT_S 中的 bClutReload 项，表明是否需要硬件重新加载 CLUT 表。在第一次作颜色扩展操作（源为 Clut 格式，目的为 ARGB/AYCbCr 格式）时，需要打开 Clut Reload 标记。

- 输出图像的剪切功能，即 clip 功能

经过 TDE 处理的图像直接输出到目的位图的指定区域。而 clip 功能则可以在输出图像时，指定输出其中的一部分到目的位图，即对输出结果做了裁剪后才输出。clip 支持两种裁剪模式：

- 区域内裁剪：指仅更新 clip 指定范围内的区域为 TDE 运算结果。如图 2-5 所示，clip 区域与目的位图的操作区域相交，区域内裁剪就导致仅将灰色区域更新为 TDE 运算结果，目标操作区域的其它地方保持不变。
- 区域外裁剪：指 TDE 运算结果仅更新 clip 指定范围外的区域。如图 2-6 所示，clip 区域与目的位图的操作区域相交，区域外裁剪就导致仅将灰色区域更新为 TDE 运算结果，clip 区域内部的地方保持不变。

图2-5 区域内 clip 示意图

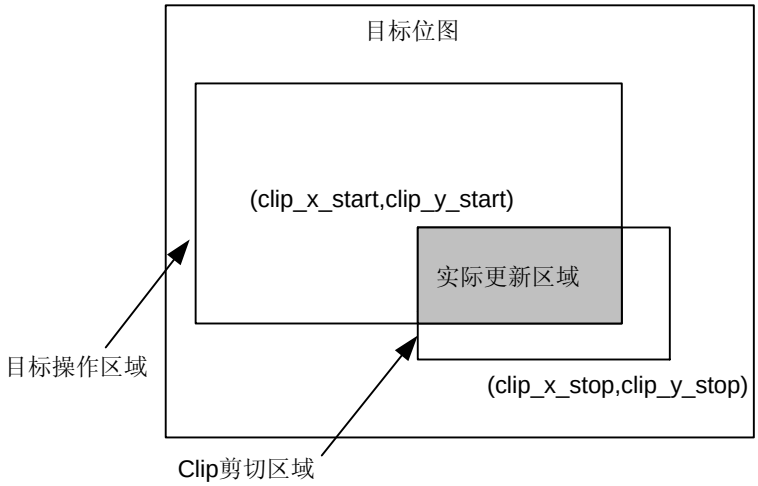
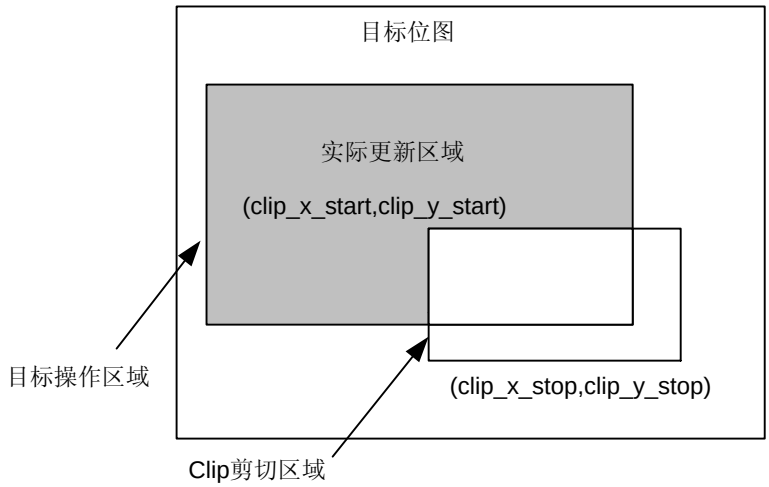


图2-6 区域外 clip 示意图



- 输出 Alpha 来源

  有 4 种模式供选择：

  - 来源于运算结果

- 来源于前景位图
- 来源于背景位图
- 来源于全局 Alpha

说明

对于 alpha 叠加运算，则需要选择来源于运算结果模式。

- 单源或双源的图形操作

  单源操作指只有一个位图来源（如仅指定背景位图和目的位图，前景位图为 NULL），针对该位图可以做以下处理：

  - 位图搬移
  - 位图格式转换
  - 位图缩放
  - 位图抗闪烁
  - 位图颜色扩展或颜色校正
  - 位图输出结果裁减，即 clip

  双源操作指有两个位图来源（背景位图和前景位图），两个位图的运算结果输出到目的位图指定的区域。其中，背景位图可以与目标位图为同一位图，此操作的含义为：将前景位图与背景位图进行运算，将结果直接输出覆盖到背景位图中。双源类的操作包括以下处理：

  - 前景和背景的 ROP 操作
  - 前景和背景的 alpha 叠加操作
  - ColorKey 操作
  - 前景位图指定区域缩放/抗闪烁处理后，再与背景做 alpha 叠加等操作

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstBackGround	背景位图。	输入	局部
pstBackGroundRect	背景位图操作区域。	输入	局部
pstForeGround	前景位图。	输入	局部
pstForeGroundRect	前景位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部
pstOpt	运算参数设置结构。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_ERR_TDE_NOT_ALIGNED	Clut 表的起始地址没有按照 4byte 对齐。
HI_ERR_TDE_MINIFICATION	缩小倍数过大。
HI_ERR_TDE_UNSUPPORTED_OPERATION	位图的格式操作不支持此操作。
HI_ERR_TDE_CLIP_AREA	操作区域与 clip 区域没有交集，显示不会有更新。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 目标位图必须与背景位图的颜色空间一致，前景位图的颜色空间可以与背景/目标位图不一致，这种情况下会进行颜色空间转换功能。
- 当前景源位图与目标位图尺寸不一致时，如果设置了缩放则按照设定的区域进行缩放，否则按照设置公共区域的最小值进行裁减搬移。
- Global Alpha 和 Alplh0、Alpha1 的设置值统一按照[0, 255]的范围进行设置。
- 背景位图可以与目标位图为同一位图。
- 当只需要使用单源搬移操作时（比如只对源位图进行 ROP 取非操作），可以将背景或背景位图的结构信息和操作区域结构指针设置为空。

- 启用镜像的同时禁止进行缩放。
- 作 clip 操作时，裁减区域必须与操作区域有公共交集，否则会返回错误。
- 在第一次作颜色扩展操作（源为 Clut 格式，目的为 ARGB/AYCbCr 格式）时，需要打开 Clut Reload 标记。
- ROP 操作时，通过操作结构体 TDE2_OPT_S 中的成员 enRopCode_Color 和 enRopCode_Alpha 分别指定颜色和 alpha 分量进行的 ROP 操作。其中，ROP 操作类型中的 S1 指背景位图 pstForeGround，S2 指前景位图 pstBackGround。

【举例】

无。

HI_TDE2_MbBlit

【目的】

向任务中添加对宏块位图进行有附加功能的搬移操作。将亮度和色度宏块数据合并成光栅格式，可以伴随缩放、抗闪烁、Clip 处理。

【语法】

```
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);
```

【描述】

宏块 surface 指定区域的亮度和色度数据合并成光栅格式输出到目标 surface 的指定区域。在合并的过程中可以伴随缩放操作，由 pstMbOpt 的 enResize 参数指定。如果没有指定缩放，将直接将宏块数据合并的结果输出到目标 surface 上，超出的部分将剪切掉。当 clip 开关打开时，将做剪切拷贝；合并过程中也支持抗闪烁处理。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstMB	宏块 surface。	输入	局部
pstMbRect	宏块操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标操作区域。	输入	局部
pstMbOpt	宏块操作属性。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_ERR_TDE_MINIFICATION	缩小倍数超出限制（最大为缩小 15 倍）。
HI_ERR_TDE_UNSUPPORTED_OPERATION	位图的格式操作不支持此操作。
HI_ERR_TDE_CLIP_AREA	操作区域与 clip 区域没有交集，显示不会有更新。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 对于 YCbCr422 格式的宏块，操作区域起始点水平坐标必须是偶数。
- 目标位图必须是 YCbCr 颜色空间。

【举例】

无。

HI_TDE2_SolidDraw

【目的】

向任务中添加对光栅位图进行有附加操作的填充搬移操作。实现在 surface 上画点、画线、色块填充或内存填充等功能。

【语法】

```
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);
```

【描述】

该接口实现前景 surface 操作区域和填充色运算后输出到目标 surface 的操作区域。该运算可以是 alpha 叠加运算或 ROP 运算，中间可以伴随着 Clip 操作。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstForeGround	前景位图。	输入	局部
pstForeGroundRect	前景位图的操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图的操作区域。	输入	局部
pstFillColor	填充色结构体。	输入	局部
pstOpt	操作属性结构体。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。

返回值	描述
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_ERR_TDE_NOT_ALIGNED	Clut 表的起始地址没有按照 4byte 对齐。
HI_ERR_TDE_MINIFICATION	缩小倍数过大。
HI_ERR_TDE_UNSUPPORTED_OPERATION	位图的格式操作不支持此操作。
HI_ERR_TDE_CLIP_AREA	操作区域与 clip 区域没有交集，显示不会有更新。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 当前景位图为 NULL 且操作结构 pstOpt 为 NULL 时，该接口可实现单纯的色彩填充功能，即与 HI_TDE2_QuickFill 实现的功能一样。接口调用形式如下：

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

- 当前景位图为 NULL，而操作结构 pstOpt 指定 alpha 叠加或 ROP 操作时，该接口可实现：目的位图（pstDst）的指定区域与填充色做 alpha 叠加或 Rop 运算，结果输出到目的位图的功能。即在单纯颜色填充基础上，可实现填充色和位图的透明叠加或 ROP 效果。接口调用形式如下：

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

  其中，pstOpt 可指定 alpha 或 ROP 操作及输出结果的剪切操作（clip），但不支持 colorkey、mirror 操作。当指定 ROP 操作时，ROP 操作对象 S1 指目的位图（pstDst），S2 指填充色。

- 当前景位图不为 NULL 时（此时操作属性 pstOpt 一定不能为 NULL），可实现前景位图的指定区域在缩放/抗闪后再与填充色做 alpha 叠加或 ROP 等操作，结果输出到目的位图的指定区域。接口调用形式如下：

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

此时，调用者可将填充色看作是背景位图（一幅颜色一致的位图），前景和背景位图可作 pstOpt 中指定的所有操作，包括前景的缩放、抗闪、前景和背景的 alpha 叠加或 ROP、前景的 colorkey、输出结果的镜像 mirror 和剪切 clip。

- 当指定 ROP 操作时，ROP 操作对象 S1 指填充色，S2 指前景位图。
- 当指定 colorkey 操作时，只能对前景做 colorkey。

- Solid Draw 绘制矩形或者水平/垂直直线的方法是通过设置填充矩形的宽/高来完成。例如：垂直直线就是绘制宽度为 1 像素的矩形。

【举例】

无。

HI_TDE2_BitmapMaskRop

【目的】

向任务中添加对光栅位图进行 Mask Rop 搬移操作。根据 Mask 位图实现前景位图和背景位图 ROP 的效果。

【语法】

```
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);
```

【描述】

Mask 位图必须为 A1 位图，在 Mask 位图为 0 的地方输出背景像素值，为 1 的地方输出前景和背景的 ROP 运算结果值。

MaskRop 与普通的 Rop 操作的不同之处有以下两点：

- 普通的 ROP 操作是两幅图像的操作区域中的每个像素点都参与 ROP 操作，无法实现部分区域做 ROP 操作，部分不做（保留背景）。
- MaksRop 操作通过构造合适的 Mask 位图可实现：输出图像的部分区域是前背景的 ROP 结果，部分区域是背景图象，就像是对前背景 ROP 的结果做了一个 clip 剪切。通过构造 Mask 位图，还可实现随即形状的 clip 剪切。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstBackGround	背景位图。	输入	局部
pstBackGroundRect	背景位图操作区域。	输入	局部
pstForeGround	前景位图。	输入	局部
pstForeGroundRect	前景位图操作区域。	输入	局部
pstMask	Mask 位图。	输入	局部
pstMaskRect	Mask 位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部
enRopCode_Color	颜色分量的 ROP 运算码。	输入	局部
enRopCode_Alpha	Alpha 分量的 ROP 运算码。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_stb_api.h

- 库文件：libhimpi.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 前景位图、背景位图、mask 位图、目的位图分别和其操作区域求得有效操作区域，4 个有效操作区域的大小必须一致。
- Mask 位图必须是 A1 格式的位图。
- 目标位图和背景位图必须位于同一颜色空间。

说明

有效操作区域：指调用者指定的操作区域与位图的相交部分。

【举例】

无。

HI_TDE_BitmapMaskBlend

【目的】

向任务中添加对光栅位图进行 Mask Blend 搬移操作。根据 Mask 位图实现前景位图和背景位图带 Mask 位图的叠加效果。

【语法】

```
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);
```

【描述】

Mask 是 A1 或 A8 的位图，在 Mask 位图为 0 的地方输出背景像素值，为 1 的地方输出前景和背景的 blending 叠加结果。

MaskBlending 与普通的 blending 叠加操作的不同之处有以下两点：

- 普通的 blending 叠加操作是两幅图像的操作区域中的每个像素点都参与叠加，无法实现部分区域做 Blending 操作，部分不做（保留背景）。
- MaksBlending 操作通过构造合适的 Mask 位图可实现：输出图像的部分区域是前背景的叠加结果，部分区域是背景图象，就像是对前背景 Blending 的结果做了一个 clip 剪切。通过构造 Mask 位图，还可实现随即形状的 clip 剪切。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部
pstBackGround	背景位图。	输入	局部
pstBackGroundRect	背景位图操作区域。	输入	局部
pstForeGround	前景位图。	输入	局部
pstForeGroundRect	前景位图操作区域。	输入	局部
pstMask	Mask 位图。	输入	局部
pstMaskRect	Mask 位图操作区域。	输入	局部
pstDst	目标位图。	输入	局部
pstDstRect	目标位图操作区域。	输入	局部
u8Alpha	Alpha 叠加时的全局 alpha 值。	输入	局部
enBlendMode	Alpha 叠加模式选择。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_NULL_PTR	参数中有空指针错误。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。
HI_ERR_TDE_INVALID_PARA	无效的参数设置。
HI_ERR_TDE_NO_MEM	内存不足，无法添加操作。
HI_FAILURE	系统错误或未知错误。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 目标位图和背景位图必须位于同一颜色空间。
- Hi3520 芯片不支持预乘模式，故前景位图应是非预乘的数据，并选择非预乘模式。
- enBlendMode 不能选择 TDE2_ALUCMD_ROP 模式。
- 前景位图、背景位图、mask 位图、目的位图分别和其操作区域求得有效操作区域，4 个有效操作区域的大小必须一致。

【举例】

无。

HI_TDE2_CancelJob

【目的】

取消 TDE 任务及已经成功加入到该任务中的操作。

【语法】

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

【描述】

向 TDE 任务添加操作时，如果出现当前的操作参数非法等错误，程序需要返回退出时，可调用此接口取消当前任务及其下的所有操作。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_FAILURE	指定的任务已经提交无法取消。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 在调用此接口前应保证调用 HI_TDE2_Open 打开 TDE 设备，并且调用 HI_TDE2_BeginJob 获得了有效的任务句柄。
- 已经提交的任务不能够再取消。
- 取消后的任务不再有效，故不能再向其添加操作，也不能提交该任务。
- 在向 TDE 任务中添加操作（如操作 A）时出错可以有以下两种处理方式：
  - 忽略出错的操作 A，继续向 TDE 任务中添加其余命令，并提交该任务。若该任务成功执行，则说明所有成功添加的操作都完成了，A 操作因未添加成功而没有执行。
  - 因添加操作 A 出错而取消整个任务，则说明该任务连同其下所有已成功添加的操作都被取消。

【举例】

```
/* 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 serival 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

【目的】

等待指定的任务完成。

【语法】

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

【描述】

当使用非阻塞方式提交 TDE 任务后，可以使用此接口等待指定的 TDE 任务完成。此接口为阻塞调用。

该接口一般用于 TDE 对一块显存进行异步（即非阻塞式）操作后，软件再对该显存进行操作，这样就存在前面的 TDE 和软件同时操作同一块显存的风险。这时，用户可以先调用此接口确保之前的 TDE 任务已经完成，然后再进行软件的操作。

【参数】

参数名称	描述	输入/输出	全局/局部
s32Handle	TDE 任务句柄。	输入	局部

【返回值】

返回值	描述
0	指定的 TDE 任务未完成。
非 0	失败，其值为错误码。

【错误码】

返回值	描述
HI_ERR_TDE_DEV_NOT_OPEN	TDE 设备未打开，API 调用失败。
HI_ERR_TDE_INVALID_HANDLE	非法的任务句柄。

返回值	描述
HI_ERR_TDE_QUERY_TIMEOUT	指定的任务超时未完成。
HI_ERR_TDE_UNSUPPORTED_OPERATION	不支持的操作。

【需求】

- 头文件：hi_tde_api.h
- 库文件：libtde.a

【注意】

- 此接口为阻塞接口，会阻塞等待指定的任务完成。
- 不允许等待一个未提交的任务，否则返回错误码 HI_ERR_TDE_UNSUPPORTED_OPERATION

【举例】

无。

3 数据类型

3.1 映射表

本章详细描述了 API 中涉及的数据结构，如表 3-1 所示。

表3-1 TDE 数据结构映射表

函数	说明
TDE_HANDLE	TDE 任务句柄
TDE2_COLOR_FMT_E	TDE 支持的光栅像素格式
TDE2_RECT_S	操作区域属性
TDE2_ALUCMD_E	TDE 逻辑运算类型
TDE_ROP_CODE_E	TDE 支持的 ROP 操作类型
TDE2_COLORKEY_MODE_E	Colorkey 模式
TDE2_COLORKEY_COMP_S	单个颜色分量的关键色属性
TDE2_COLORKEY_U	Colorkey 关键色属性
TDE2_CLIPMODE_E	剪切操作类型
TDE2_OUTALPHA_FROM_E	输出 alpha 来源类型
TDE2_FILTER_MODE_E	图像滤波模式
TDE2_MIRROR_E	图像镜像属性
TDE2_SURFACE_S	位图 surface 结构体
TDE2_OPT_S	TDE 操作属性结构体
TDE2_MB_COLOR_FMT_E	TDE 支持的宏块格式
TDE_PIC_MODE_E	帧场处理模式
TDE2_MB_S	宏块格式位图基本属性

3.2 详细描述

3.2.1 TDE_HANDLE

【说明】

TDE 任务句柄。

【定义】

```
typedef HI_S32 TDE_HANDLE;
```

【成员】

无。

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.2 TDE2_COLOR_FMT_E

【说明】

TDE 支持的像素格式。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_COLOR_FMT_RGB444	RGB444 格式
TDE2_COLOR_FMT_RGB555	ARGB555 格式
TDE2_COLOR_FMT_RGB565	RGB565 格式
TDE2_COLOR_FMT_RGB888	RGB888 格式
TDE2_COLOR_FMT_ARGB4444	ARGB4444 格式
TDE2_COLOR_FMT_ARGB1555	ARGB1555 格式
TDE2_COLOR_FMT_ARGB8565	ARGB8565 格式
TDE2_COLOR_FMT_ARGB8888	ARGB8888 格式
TDE2_COLOR_FMT_CLUT1	CLUT1 格式
TDE2_COLOR_FMT_CLUT4	CLUT4 格式
TDE2_COLOR_FMT_CLUT8	CLUT8 格式
TDE2_COLOR_FMT_ACLUT44	ACLUT44 格式
TDE2_COLOR_FMT_ACLUT88	ACLUT88 格式
TDE2_COLOR_FMT_A1	A1 格式
TDE2_COLOR_FMT_A8	A8 格式
TDE2_COLOR_FMT_YCbCr888	YCbCr888 格式
TDE2_COLOR_FMT_AYCbCr8888	AYCbCr8888 格式
TDE2_COLOR_FMT_YCbCr422	YCbCr422 格式
TDE2_COLOR_FMT_byte	byte 格式
TDE2_COLOR_FMT_halfword	halfword 格式
TDE_COLOR_FMT_BUTT	无效的像素格式

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.3 TDE2_RECT_S

【说明】

TDE 操作区域属性。

【定义】

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

【成员】

成员	描述
s32Xpos	操作区域的起始横坐标，以像素数为单位。 有效范围：[0, 位图宽度）。
s32Ypos	操作区域的起始纵坐标，以像素数为单位。 有效范围：[0, 位图高度）。
u32Width	操作区域的宽度，以像素数为单位。 有效范围：(0, 0xFFF]。
u32Height	操作区域的高度，以像素数为单位。 有效范围：(0, 0xFFF]。

【注意事项】

- 操作区域与位图的关系如图 2-1 所示。
- 若操作区域与位图部分相交，则取相交部分为实际参与操作的区域；若操作区域与位图不相交，则返回相应错误码。

【相关数据类型和接口】

无。

3.2.4 TDE2_ALUCMD_E

【说明】

逻辑运算类型属性。

【定义】

```
typedef enum hiTDE2_ALUCMD_E
{
    TDE2_ALUCMD_NONE = 0,
    TDE2_ALUCMD_BLEND,
    TDE2_ALUCMD_BLEND_PREMUL,
    TDE2_ALUCMD_ROP,
    TDE2_ALUCMD_BUTT
} TDE2_ALUCMD_E;
```

【成员】

成员	描述
TDE2_ALUCMD_NONE	无逻辑运算
TDE2_ALUCMD_BLEND	Alpha 叠加类型
TDE2_ALUCMD_BLEND_PREMUL	预乘的 alpha 叠加类型，Hi3520 暂不支持
TDE2_ALUCMD_ROP	布尔运算类型
TDE2_ALUCMD_BUTT	无效逻辑运算类型

【注意事项】

- 若进行两个位图的 alpha 叠加运算，请选择 TDE2_ALUCMD_BLEND，参与运算的位图必须为非 alpha 预乘的位图。Hi3520 芯片不支持预乘形式的 alpha 叠加。
- 如选择 TDE2_ALUCMD_ROP，指进行布尔逻辑运算类型。通过指定 TDE2_OPT_S 结构中的 enRopCode_Color 和 enRopCode_Alpha 成员，分别指定颜色和 alpha 分量的 ROP 运算类型。

【相关数据类型和接口】

无。

3.2.5 TDE_ROP_CODE_E

【说明】

TDE 支持的 ROP 操作类型。

【定义】

```
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;
```

【成员】

成员	描述
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
TDE2_ROP_WHITE	Whiteness
TDE_ROP_BUTT	无效的 ROP 类型

注 a：S1 表示位图 1，S2 表示位图 2。

【注意事项】

不同操作时，S1、S2 具体所指的位图不同，详见每个接口的说明部分。

【相关数据类型和接口】

无。

3.2.6 TDE2_COLORKEY_MODE_E

【说明】

TDE colorkey 模式属性。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_COLORKEY_MODE_NONE	不做 colorkey 操作
TDE2_COLORKEY_MODE_FOREGROUND	对前景位图进行 colorkey 操作
TDE2_COLORKEY_MODE_BACKGROUND	对背景位图进行 colorkey 操作
TDE2_COLORKEY_MODE_BUTT	无效的 colorkey 模式

【注意事项】

对前景位图进行 colorkey 时，对于颜色扩展，在 CLUT 前做 colorkey；对于颜色校正，在 CLUT 后做 colorkey。

【相关数据类型和接口】

无。

3.2.7 TDE2_COLORKEY_COMP_S

【说明】

单个颜色分量的关键色属性。

【定义】

```
typedef struct hiTDE2_COLORKEY_COMP_S
{
    HI_U8 u8CompMin;          /* 分量最小值*/
    HI_U8 u8CompMax;          /* 分量最大值*/
    HI_U8 bCompOut;           /* 分量关键色在范围内或范围外*/
    HI_U8 bCompIgnore;        /* 分量是否忽略*/
} TDE2_COLORKEY_COMP_S;
```

【成员】

成员	描述
u8CompMin	分量关键色最小值
u8CompMax	分量关键色最大值
bCompOut	分量关键色在范围内或范围外
bCompIgnore	分量是否忽略

【注意事项】

结构成员 bCompIgnore 指关键色比较时是否忽略该分量的比较，而总是认为其满足关键色要求。

- 若 bCompIgnore 为 TRUE，表示关键色比较时，忽略该分量的比较，认为该分量总是满足关键色要求。
- 若 bCompIgnore 为 FALSE，表示需要根据[最小关键色, 最大关键色]范围以及属性 bCompOut 判断该分量的值是否符合关键色要求。

【相关数据类型和接口】

无。

3.2.8 TDE2_COLORKEY_U

【说明】

关键色属性。

【定义】

```
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;
```

【成员】

结构 struCkARGB 成员：表示位图格式为 ARGB 类型时各分量的关键色属性。

成员	描述
stAlpha	alpha 分量关键色属性
stRed	Red 分量关键色属性
stGreen	Green 分量关键色属性
stBlue	Blue 分量关键色属性

结构 struCkYCbCr 成员：表示位图格式为 AYCbCr 类型时，各分量的关键色属性。

成员	描述
stAlpha	alpha 分量关键色属性
stY	Y 分量关键色属性
stCb	Cb 分量关键色属性
stCr	Cr 分量关键色属性

结构 struCkClut 成员：表示位图格式为 CLUT 类型时，各分量的关键色属性。

成员	描述
stAlpha	alpha 分量关键色属性
stClut	CLUT 分量关键色属性

联合类型 TDE2_COLORKEY_U 成员，各分量的关键色属性。

成员	描述
struCkARGB	当位图格式为 ARGB 类型时，关键色属性
struCkYCbCr	当位图格式为 AYCbCr 类型时，关键色属性
struCkClut	当位图格式为 CLUT 类型时，关键色属性

【注意事项】

- 当 bColorSpace 为 HI_TRUE 时，color space 使能。此时 color space 对象颜色值的各分量都落在 stColorMin 和 stColorMax 对应分量之间的像素点不参与运算。详细描述请参见芯片手册中关于 TDE 模块的说明。
- 不管当前位图是什么格式，color space 颜色上限值和下限值都应该是 ARGB8888 格式的值。

【相关数据类型和接口】

无。

3.2.9 TDE2_CLIPMODE_E

【说明】

剪切操作类型。

【定义】

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

【成员】

成员	描述
TDE2_CLIPMODE_NONE	输出结果不做剪切
TDE2_CLIPMODE_INSIDE	区域内剪切模式
TDE2_CLIPMODE_OUTSIDE	区域外剪切模式
TDE2_CLIPMODE_BUTT	无效的剪切模式

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.10 TDE2_OUTALPHA_FROM_E

【说明】

输出 alpha 来源类型。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_OUTALPHA_FROM_NORM	输出图像的 alpha 来源于 alpha blending 的结果或者抗闪烁的结果
TDE2_OUTALPHA_FROM_BACKGROUND	输出图像的 alpha 来源于背景位图
TDE2_OUTALPHA_FROM_FOREGROUND	输出图像的 alpha 来源于前景位图
TDE2_OUTALPHA_FROM_GLOBALALPHA	输出图像的 alpha 来源于全局 alpha

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.11 TDE2_FILTER_MODE_E

【说明】

图像滤波模式。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_FILTER_MODE_COLOR	对颜色进行滤波
TDE2_FILTER_MODE_ALPHA	对 alpha 通道滤波
TDE2_FILTER_MODE_BOTH	对颜色和 alpha 通道同时滤波
TDE2_FILTER_MODE_BUTT	无效的滤波模式

【注意事项】

图像的缩放或抗闪都是一种滤波，故在图像缩放或（和）抗闪操作时，需要指明滤波模式。

【相关数据类型和接口】

无。

3.2.12 TDE2_FILLCOLOR_S

【说明】

图像填充色属性结构。

【定义】

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

【成员】

成员	描述
enColorFmt	填充色格式
u32FillColor	填充值

【注意事项】

填充值必须与填充格式相匹配，如某位图需填充为蓝色，可指定填充色格式为 ARGB1555，填充值为 0x801F（此处 alpha 位为 1）。

【相关数据类型和接口】

无。

3.2.13 TDE2_MIRROR_E

【说明】

图像镜像属性。

【定义】

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

【成员】

成员	描述
TDE2_MIRROR_NONE	输出图像不进行镜像操作
TDE2_MIRROR_HORIZONTAL	输出图像水平镜像
TDE2_MIRROR_VERTICAL	输出图像垂直镜像
TDE2_MIRROR_BOTH	输出图像水平+垂直镜像
TDE2_MIRROR_BUTT	无效的镜像类型

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.14 TDE2_SURFACE_S

【说明】

位图 surface 结构体。

【定义】

```
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;
```

【成员】

成员	描述
u32PhyAddr	位图首地址。
enColorFmt	位图格式。
u16Height	位图高度。
u16Width	位图宽度。
u16Stride	位图跨度。
pu8ClutPhyAddr	Clut 表首地址，用作颜色扩展或颜色校正。
bYCbCrClut	Clut 表是否位于 YCbCr 空间。
bAlphaMax255	位图 alpha 最大值为 255 还是 128。
bAlphaExt1555	是否使能 1555 的 Alpha 扩展。 当位图格式为 ARGB1555 时，该项有效。
u8Alpha0	Alpha0 值。 取值范围：[0, 55]。 当位图格式为 ARGB1555 且 bAlphaExt1555 为 TRUE 时，该项有效。 在 ARGB1555 格式下，当像素的最高位为 0 时，选择该值作为 alpha 叠加的 alpha 值。

成员	描述
u8Alpha1	Alpha1 值。 取值范围：[0, 255]。 当位图格式为 ARGB1555 且 bAlphaExt1555 为 TRUE 时，该项有效。 在 ARGB1555 格式下，当像素的最高位为 1 时，选择该值作为 alpha 叠加的 alpha 值。

【注意事项】

- 像素格式大于等于 Byte 的位图格式的位图首地址和 Stride 必须按照像素格式对齐，像素格式不足 Byte 的位图格式的位图首地址和 Stride 需要按照 Byte 对齐。
- 像素格式不足 Byte 的位图格式的水平起始位置和宽度必须按照像素对齐。
- YCbCr422 格式的位图的水平起始位置和宽度必须为偶数。
- CLUT 到真彩色 ARGB 的扩展是依靠检索 CLUT 表来实现的。故颜色扩展功能（如 CLUT1 格式位图扩展到 ARGB8888 格式位图）或颜色校正时，需要配置 Clut 表首地址 pu8ClutPhyAddr，且其指向的 CLUT 表内存必须是物理连续的。

【相关数据类型和接口】

无。

3.2.15 TDE2_OPT_S

【说明】

TDE 操作属性结构体。

【定义】

```
typedef struct hiTDE2_OPT_S
{
    TDE2_ALUCMD_E enAluCmd;                 /*逻辑运算类型*/
    TDE2_ROP_CODE_E enRopCode_Color;        /*颜色空间ROP类型*/
    TDE2_ROP_CODE_E enRopCode_Alpha;        /*Alpha的ROP类型*/
    TDE2_COLORKEY_MODE_E enColorKeyMode;    /*colorkey方式*/
    TDE2_COLORKEY_U unColorKeyValue;        /*colorkey设置值*/
    TDE2_CLIPMODE_E enClipMode;             /*区域内作clip还是区域外作clip*/
    TDE2_RECT_S stClipRect;                 /*clip区域定义*/
    HI_BOOL bDeflicker;                     /*是否抗闪烁*/
    HI_BOOL bResize;                        /*是否缩放*/
    TDE2_FILTER_MODE_E enFilterMode;        /*缩放或deflicker时使用的滤波模式
*/
    TDE2_MIRROR_E enMirror;                 /*镜像类型*/
```

```
    HI_BOOL bClutReload;                    /*是否重新加载Clut表*/
    HI_U8   u8GlobalAlpha;                  /*全局alpha值*/
    TDE2_OUTALPHA_FROM_E enOutAlphaFrom;    /*输出alpha来源*/
} TDE2_OPT_S
```

【成员】

成员	描述
enAluCmd	逻辑运算类型
enRopCode_Color	颜色空间 ROP 类型
enRopCode_Alpha	Alpha 的 ROP 类型
enColorKeyMode	colorkey 方式
unColorKeyValue	colorkey 设置值
enClipMode	区域内作 clip 还是区域外作 clip
stClipRect	clip 区域定义
bDeflicker	是否抗闪烁
bResize	是否缩放
enFilterMode	缩放或 deflicker 时使用的滤波模式
enMirror	镜像类型
bClutReload	是否重新加载 Clut 表
u8GlobalAlpha	全局 alpha 值 取值范围：[0, 255]
enOutAlphaFrom	输出 alpha 来源

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.16 TDE2_MB_COLOR_FMT_E

【说明】

TDE 支持的宏块格式类型。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_MB_COLOR_FMT_JPG_YCbCr400MBP	JPEG 编码 400 宏块
TDE2_MB_COLOR_FMT_JPG_YCbCr422MBHP	JPEG 编码 422 宏块（水平采样一半）
TDE2_MB_COLOR_FMT_JPG_YCbCr422MBVP	JPEG 编码 422 宏块（垂直采样一半）
TDE2_MB_COLOR_FMT_MP1_YCbCr420MBP	MPEG-1 编码 420 宏块
TDE2_MB_COLOR_FMT_MP2_YCbCr420MBP	MPEG-2 编码 420 宏块
TDE2_MB_COLOR_FMT_MP2_YCbCr420MBI	MPEG-2 编码 420 宏块（隔行）
TDE2_MB_COLOR_FMT_JPG_YCbCr420MBP	JPEG 编码 420 宏块
TDE2_MB_COLOR_FMT_JPG_YCbCr444MBP	JPEG 编码 444 宏块

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.17 TDE_PIC_MODE_E

【说明】

TDE 支持的帧场处理模式。

【定义】

```
typedef enum hiTDE_PIC_MODE_E
```

```
{
    TDE_FRAME_PIC_MODE = 0,      /*帧处理模式*/
    TDE_BOTTOM_FIELD_PIC_MODE, /*底场处理模式*/
    TDE_TOP_FIELD_PIC_MODE,      /*顶场处理模式*/
    TDE_TB_FIELD_PIC_MODE,       /*顶底场处理模式*/
    TDE_PIC_MODE_BUTT
} TDE_PIC_MODE_E;
```

【成员】

成员	描述
TDE_FRAME_PIC_MODE	帧处理模式
TDE_BOTTOM_FIELD_PIC_MODE	底场处理模式
TDE_TOP_FIELD_PIC_MODE	顶场处理模式
TDE_TB_FIELD_PIC_MODE	顶底场处理模式
TDE_PIC_MODE_BUTT	无效的帧场处理模式

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.18 TDE2_MB_S

【说明】

宏块 Surface 结构体，描述宏块格式的图像的基本信息。

【定义】

```
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;  /*颜色空间转换矩阵*/
    TDE_PIC_MODE_E       enPicMode;
```

```
} TDE2_MB_S;
```

【成员】

成员	描述
enMbFmt	宏块格式
u32YPhyAddr	亮度块物理首地址
u32YWidth	亮度块的宽度
u32YHeight	亮度块的高度
u32YStride	亮度块相邻两行的跨度
u32CbCrPhyAddr	色度块的物理首地址
u32CbCrStride	色度块相邻两行的跨度
enColorSpaceConv	颜色空间转换矩阵类型
enPicMode	位图的帧场处理模式

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.19 TDE2_COLORSPACE_CONV_MODE_E

【说明】

宏块格式转换为光栅格式时的颜色空间转换模式。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_ITU_R_BT601_IMAGE	转化时采用 BT601 标准的图像转换标准

成员	描述
TDE2_ITU_R_BT709_IMAGE	转化时采用 BT709 标准的图像转换标准
TDE2_ITU_R_BT601_VIDEO	转化时采用 BT601 标准的视频转换标准
TDE2_ITU_R_BT709_VIDEO	转化时采用 BT709 标准的视频转换标准

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.20 TDE2_MBRESIZE_E

【说明】

宏块格式缩放类型。

【定义】

```
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;
```

【成员】

成员	描述
TDE2_MBRESIZE_NONE	不进行缩放
TDE2_MBRESIZE_QUALITY_LOW	宏块 Suface 的低质量缩放模式
TDE2_MBRESIZE_QUALITY_MIDDLE	宏块 Suface 的中质量缩放模式
TDE2_MBRESIZE_QUALITY_HIGH	宏块 Suface 的高质量缩放模式

【注意事项】

无。

【相关数据类型和接口】

无。

3.2.21 TDE2_MBOPT_S

【说明】

宏块 Surface 的操作属性。

【定义】

```
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_MBOPT_S;
```

【成员】

成员	描述
enClipMode	Clip 模式选择：区域内作 clip 还是区域外作 clip
stClipRect	Clip 区域定义
bDeflicker	是否抗闪烁
enResize	宏块缩放模式：不缩放/高质量缩放/中质量缩放/低质量缩放
bSetOutAlpha	是否用户指定输出结果位图的 alpha 值 如果不设置 Alpha，则默认输出最大 Alpha 值
u8OutAlpha	用户指定的输出结果位图的 alpha 值

【注意事项】

无。

【相关数据类型和接口】

无。

4 实例

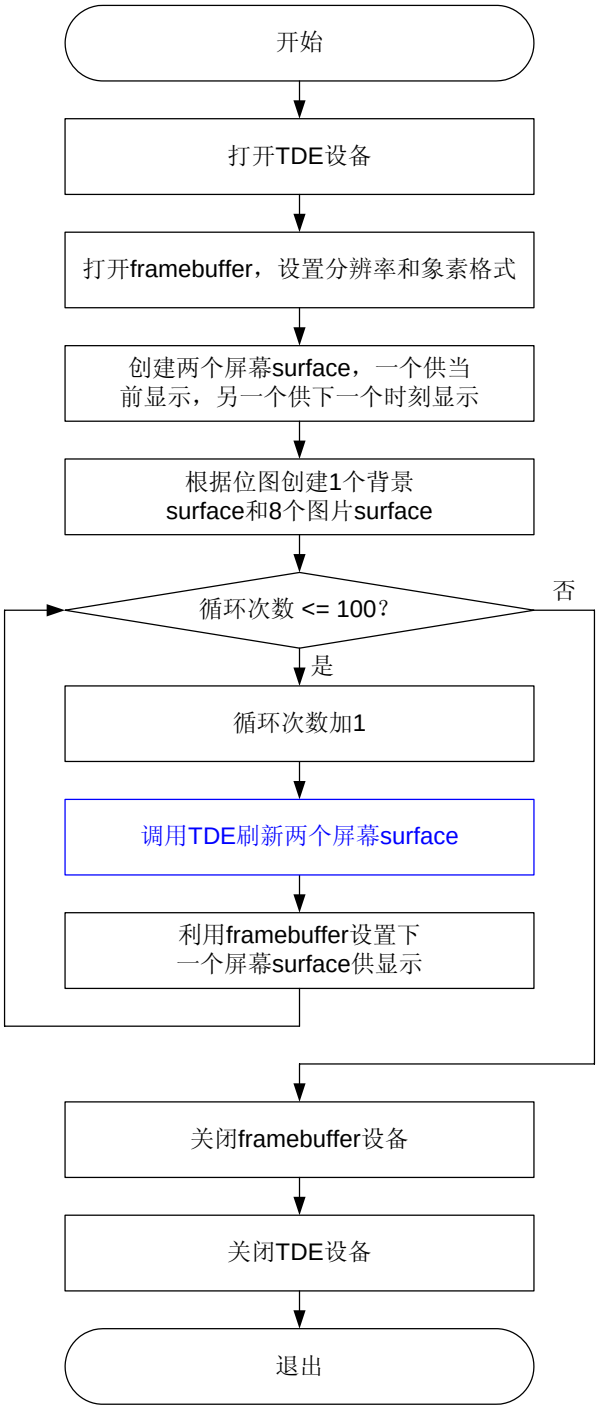
4.1 软件流程

说明

本章以利用 bitblt 和 color space 实现图片动画旋转效果为例。在应用 TDE 前应保证 TDE 和 HiFB 驱动已经加载，视频输出设备处于工作状态。运行本示例至少需要给叠加图像层 0 分配 3391488 字节的显存，有关 HiFB 的加载请参见《HiFB 开发指南》。

该例的软件实现流程如图 4-1 所示。

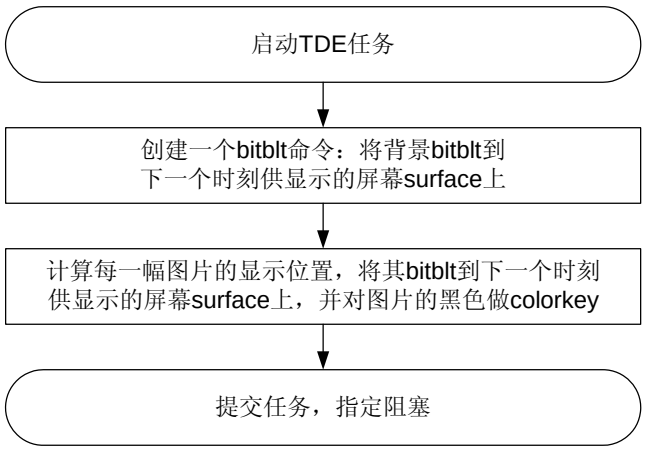
图4-1 软件实现的流程图（主流程）



注：“调用 TDE 刷新两个屏幕 surface”的详细流程请参见图 4-2。

调用 TDE 刷新两个屏幕 surface 函数的实现过程如图 4-2 所示。

图4-2 调用 TDE 刷新两个屏幕 surface 函数的实现过程



4.2 代码参考

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <fcntl.h>

#include "hi_tde_api.h"
#include "hi_tde_type.h"
#include "hi_tde_errcode.h"
#include "hifb.h"
//#include "hi_unf_avout.h"

#define TDE_PRINT printf

#define MIN(x,y) ((x) > (y) ? (y) : (x))
#define MAX(x,y) ((x) > (y) ? (x) : (y))

static const HI_CHAR *pszImageNames[] =
{
    "res/apple.bits",
    "res/applets.bits",
```

```
    "res/calendar.bits",
    "res/foot.bits",
    "res/gmush.bits",
    "res/gimp.bits",
    "res/gsame.bits",
    "res/keys.bits"
};

#define N_IMAGES (HI_S32)((sizeof (pszImageNames) / sizeof
(pszImageNames[0])))

#define BACKGROUND_NAME  "res/background.bits"

#define PIXFMT  TDE2_COLOR_FMT_ARGB8888
#define BPP     4
#define SCREEN_WIDTH    720
#define SCREEN_HEIGHT   576
#define CYCLE_LEN      60

static HI_S32   g_s32FrameNum;
static TDE2_SURFACE_S g_stScreen[2];
static TDE2_SURFACE_S g_stBackGround;
static TDE2_SURFACE_S g_stImgSur[N_IMAGES];

static HI_S32 TDE_CreateSurfaceByFile(const HI_CHAR *pszFileName,
TDE2_SURFACE_S *pstSurface, HI_U8 *pu8Virt)
{
    FILE *fp;
    HI_U32 colorfmt, w, h, stride;

    if((NULL == pszFileName) || (NULL == pstSurface))
    {
        printf("%s, LINE %d, NULL ptr!\n", __FUNCTION__, __LINE__);
        return -1;
    }

    fp = fopen(pszFileName, "rb");
    if(NULL == fp)
    {
        printf("error when open pszFileName %s, line:%d\n", pszFileName,
__LINE__);
        return -1;
    }
```

```
    fread(&colorfmt, 1, 4, fp);
    fread(&w, 1, 4, fp);
    fread(&h, 1, 4, fp);
    fread(&stride, 1, 4, fp);

    pstSurface->enColorFmt = colorfmt;
    pstSurface->u32Width = w;
    pstSurface->u32Height = h;
    pstSurface->u32Stride = stride;
    pstSurface->u8Alpha0 = 0;
    pstSurface->u8Alpha1 = 0xff;
    pstSurface->bAlphaMax255 = HI_TRUE;

    fread(pu8Virt, 1, stride*h, fp);
    fclose(fp);
    return 0;
}


static HI_VOID circumrotate (HI_U32 u32CurOnShow)
{
    TDE_HANDLE s32Handle;
    TDE2_OPT_S stOpt = {0};
    HI_FLOAT eXMid, eYMid;
    HI_FLOAT eRadius;
    HI_U32 i;
    HI_FLOAT f;
    HI_U32 u32NextOnShow;
    TDE2_RECT_S stSrcRect;
    TDE2_RECT_S stDstRect;
    HI_S32 s32Ret;

    u32NextOnShow = !u32CurOnShow;

    stOpt.enOutAlphaFrom = TDE2_OUTALPHA_FROM_FOREGROUND;
    stOpt.enColorKeyMode = TDE2_COLORKEY_MODE_FOREGROUND;
    stOpt.unColorKeyValue.struCkARGB.stAlpha.bCompIgnore = HI_TRUE;
    f = (float) (g_s32FrameNum % CYCLE_LEN) / CYCLE_LEN;

    stSrcRect.s32Xpos = 0;
    stSrcRect.s32Ypos = 0;
    stSrcRect.u32Width = g_stBackGround.u32Width;
    stSrcRect.u32Height = g_stBackGround.u32Height;
```

```
    eXMid = g_stBackGround.u32Width/2.16f;
    eYMid = g_stBackGround.u32Height/2.304f;

    eRadius = MIN (eXMid, eYMid) / 2.0f;

    /* 1. start job */
    s32Handle = HI_TDE2_BeginJob();
    if(HI_ERR_TDE_INVALID_HANDLE == s32Handle)
    {
        TDE_PRINT("start job failed!\n");
        return ;
    }

    /* 2. bitblt background to screen */
s32Ret = HI_TDE2_QuickCopy(s32Handle, &g_stBackGround, &stSrcRect,
&g_stScreen[u32NextOnShow], &stSrcRect);
    if(s32Ret < 0)
    {
        TDE_PRINT("Line:%d,HI_TDE2_QuickCopy failed,ret=0x%x!\n",
__LINE__, s32Ret);
         HI_TDE2_CancelJob(s32Handle);
        return ;
    }

    for(i = 0; i < N_IMAGES; i++)
    {
        HI_FLOAT ang;
        HI_FLOAT r;

        stSrcRect.s32Xpos = 0;
        stSrcRect.s32Ypos = 0;
        stSrcRect.u32Width = g_stImgSur[i].u32Width;
        stSrcRect.u32Height = g_stImgSur[i].u32Height;

        /* 3. calculate new pisition */
        ang = 2.0f * (HI_FLOAT) M_PI * (HI_FLOAT) i / N_IMAGES -
f * 2.0f * (HI_FLOAT) M_PI;
        r = eRadius + (eRadius / 3.0f) * sinf (f * 2.0 * M_PI);

        stDstRect.s32Xpos = eXMid + r * cosf (ang)
 - g_stImgSur[i].u32Width / 2.0f;;
        stDstRect.s32Ypos = eYMid + r * sinf (ang)
 - g_stImgSur[i].u32Height / 2.0f;
        stDstRect.u32Width = g_stImgSur[i].u32Width;
```

```
        stDstRect.u32Height = g_stImgSur[i].u32Height;

        /* 4. bitblt image to screen */
        s32Ret = HI_TDE2_Bitblit(s32Handle, &g_stScreen[u32NextOnShow],
&stDstRect, &g_stImgSur[i], &stSrcRect,
&g_stScreen[u32NextOnShow], &stDstRect, &stOpt);
        if(s32Ret < 0)
        {
                TDE_PRINT("Line:%d,HI_TDE2_Bitblit failed,ret=0x%x!\n",
 __LINE__, s32Ret);
            HI_TDE2_CancelJob(s32Handle);
            return ;
        }
    }

    /* 5. submit job */
    s32Ret = HI_TDE2_EndJob(s32Handle, HI_FALSE, HI_TRUE, 10);
    if(s32Ret < 0)
    {
        TDE_PRINT("Line:%d,HI_TDE2_EndJob failed,ret=0x%x!\n",
 __LINE__, s32Ret);
        HI_TDE2_CancelJob(s32Handle);
        return ;
    }

    g_s32FrameNum++;
    return;
}

HI_S32 main(HI_VOID)
{
    HI_U32 u32Size;
    HI_S32 s32Fd;
    HI_U32 u32Times;
    HI_U8* pu8Screen;
    HI_U32 u32PhyAddr;
    HI_S32 s32Ret = -1;
    HI_U32 i = 0;

    struct fb_fix_screeninfo stFixInfo;
    struct fb_var_screeninfo stVarInfo;
    struct fb_bitfield stR32 = {16, 8, 0};
    struct fb_bitfield stG32 = {8, 8, 0};
    struct fb_bitfield stB32 = {0, 8, 0};
```

```
    struct fb_bitfield stA32 = {24, 8, 0};

    /* 1. open tde device */
    HI_TDE2_Open();

    /* 2. framebuffer operation */
    s32Fd = open("/dev/fb0", O_RDWR);
    if (s32Fd == -1)
    {
        printf("open frame buffer device error\n");
        goto FB_OPEN_ERROR;
    }

    stVarInfo.xres_virtual      = SCREEN_WIDTH;
    stVarInfo.yres_virtual      = SCREEN_HEIGHT*2;
    stVarInfo.xres          = SCREEN_WIDTH;
    stVarInfo.yres          = SCREEN_HEIGHT;
    stVarInfo.activate      = FB_ACTIVATE_NOW;
    stVarInfo.bits_per_pixel    = 32;
    stVarInfo.xoffset = 0;
    stVarInfo.yoffset = 0;
    stVarInfo.red   = stR32;
    stVarInfo.green = stG32;
    stVarInfo.blue  = stB32;
    stVarInfo.transp = stA32;

    if (ioctl(s32Fd, FBIOPUT_VSCREENINFO, &stVarInfo) < 0)
    {
        printf("process frame buffer device error\n");
        goto FB_PROCESS_ERROR0;
    }

    if (ioctl(s32Fd, FBIOGET_FSCREENINFO, &stFixInfo) < 0)
    {
        printf("process frame buffer device error\n");
        goto FB_PROCESS_ERROR0;
    }

    u32Size = stFixInfo.smem_len;
    u32PhyAddr  = stFixInfo.smem_start;
pu8Screen   = mmap(NULL, u32Size, PROT_READ|PROT_WRITE, MAP_SHARED,
s32Fd, 0);
    memset(pu8Screen, 0xff, SCREEN_WIDTH*SCREEN_HEIGHT*4);
```

```
    /* 3. create surface */
    g_stScreen[0].enColorFmt = PIXFMT;
    g_stScreen[0].u32PhyAddr = u32PhyAddr;
    g_stScreen[0].u32Width = SCREEN_WIDTH;
    g_stScreen[0].u32Height = SCREEN_HEIGHT;
    g_stScreen[0].u32Stride = stFixInfo.line_length;
    g_stScreen[0].bAlphaMax255 = HI_TRUE;

    g_stScreen[1] = g_stScreen[0];
g_stScreen[1].u32PhyAddr = g_stScreen[0].u32PhyAddr
+ g_stScreen[0].u32Stride * g_stScreen[0].u32Height;

g_stBackGround.u32PhyAddr = g_stScreen[1].u32PhyAddr
 + g_stScreen[1].u32Stride * g_stScreen[1].u32Height;
TDE_CreateSurfaceByFile(BACKGROUND_NAME, &g_stBackGround,
 pu8Screen + ((HI_U32)g_stBackGround.u32PhyAddr - u32PhyAddr));
g_stImgSur[0].u32PhyAddr = g_stBackGround.u32PhyAddr
 + g_stBackGround.u32Stride * g_stBackGround.u32Height;
    for(i = 0; i < N_IMAGES - 1; i++)
    {
      TDE_CreateSurfaceByFile(pszImageNames[i], &g_stImgSur[i],
 pu8Screen + ((HI_U32)g_stImgSur[i].u32PhyAddr - u32PhyAddr));
      g_stImgSur[i+1].u32PhyAddr = g_stImgSur[i].u32PhyAddr
 + g_stImgSur[i].u32Stride * g_stImgSur[i].u32Height;
    }
TDE_CreateSurfaceByFile(pszImageNames[i], &g_stImgSur[i],
 pu8Screen + ((HI_U32)g_stImgSur[i].u32PhyAddr - u32PhyAddr));


    g_s32FrameNum = 0;

    /* 3. use tde and framebuffer to realize rotational effect */
    for (u32Times = 0; u32Times < 1000; u32Times++)
    {
           circumrotate(u32Times%2);
           stVarInfo.yoffset = (u32Times%2)?0:576;

           /*set frame buffer start position*/
           if (ioctl(s32Fd, FBIOPAN_DISPLAY, &stVarInfo) < 0)
           {
               TDE_PRINT("process frame buffer device error\n");
               goto FB_PROCESS_ERROR1;
           }
    }
```

```
    s32Ret = 0;
FB_PROCESS_ERROR1:
    munmap(pu8Screen, u32Size);
FB_PROCESS_ERROR0:
    close(s32Fd);
FB_OPEN_ERROR:
    HI_TDE2_Close();

    return s32Ret;
}
```