ByteSize_Per_Pixel = 1 （对于 YCbCr420）

2 （对于 16 位 RGB 和 YcbCr422）

4 （对于 24 位 RGB）

偏移地址用于以下两种情况：一种是为了放大/缩小物象，截取源图像的一些部分，如图 16-7 (a)所示；另一种是为了 PIP（picture-in-picture）应用，重新存储目标图像，如图 16-7 (b)所示。两种情况都必须满足字边界限制。

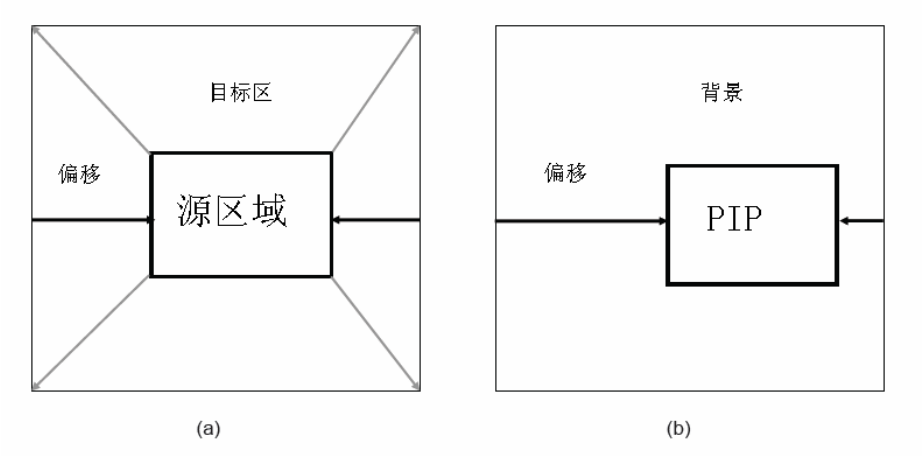

图16-7 (a)源图像放大/缩小操作的偏移地址和(b) PIP应用的目标图像

16.5 TV 定标器的帧管理

1. 每帧的管理模式

TV定标器的每帧处理被两个控制寄存器控制，像POSTENVID 和POSTINT，如图16-8所示。“POSTENVID”触发TV定标器的操作。当给予的帧完成所有操作，则它自动无效。在“POSTENVID”有效之前，所有控制寄存器必须设置为合适的值。当操作完成，中断等待寄存器有效（POSTINT=1）。如果中断有效信号有效（INTEN=1），POSTINT信号（指向中断控制器），必须通过中断服务程序清除。轮询POSTENVID也用于检测操作的结束。

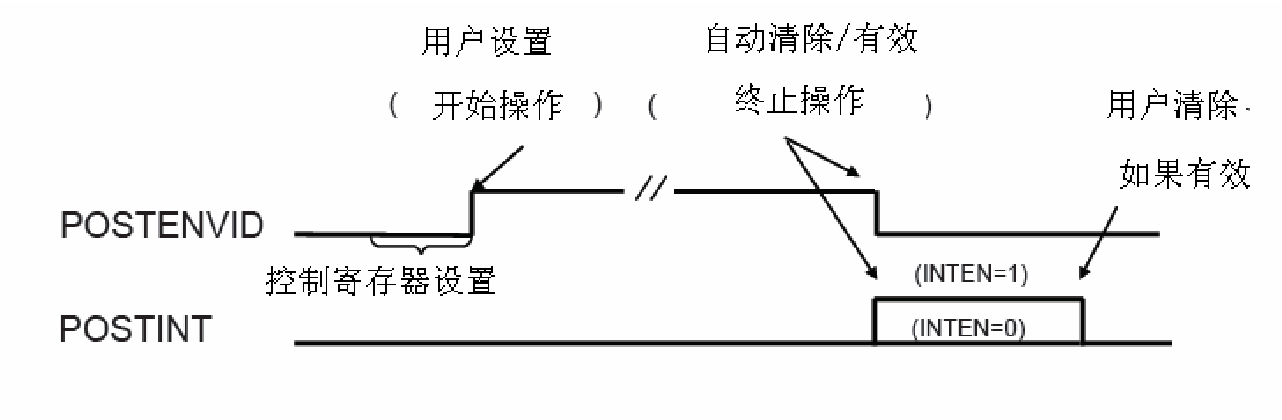

图16-8 TV定标器操作的开始和终止（AutoLoadEnable = 0）

2. **自由运行模式**

为了激活新的帧管理方式，即自运行操作，必须设置“AutoLoadEnable”为1。在该模式中，用户能预定义下一相关的帧NxtADDRXXX的地址设置。

当前帧完成时，下面的操作一步步执行：

（1）根据INTEN ，判断中断信号是否有效；

（2）NxtADDRXXX的下一帧地址设置，拷贝到ADDRXXX 的当前帧地址设置；

（3）ENVID 自动有效，并且下一帧的操作开始。

在自运行模式下，除非地址为满，否则其它寄存器的值在当前帧和下一帧必须保持相同的值。

16.6 定标器（DMA 到 FIFO）代码实现

下面是 ARM11 中，DMA 到 FIFO 部分代码的具体实现，结合上面对定标器的说明，理解以下代码：

```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////                    DMA to FIFO                    ///////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
static void ScalerT_SetCscTypeDmaToFifo(void)
{
    int nSelSrcDataFmt;
    while (1)
```

```
    {
        UART_Printf("\n");
        UART_Printf("[1] RGB16\n");
        UART_Printf("[2] RGB24\n");
        UART_Printf("[3] YCbYCr422 Interleave\n");
        UART_Printf("[4] YCrYCb422 Interleave\n");
        UART_Printf("[5] CbYCrY422 Interleave\n");
        UART_Printf("[6] CrYCbY422 Interleave\n");
        UART_Printf("[7] YUV 420 Non-Interleave\n");
        UART_Printf(">> Enter Source Data Format: ");

        nSelSrcDataFmt = UART_GetIntNum();

        if (nSelSrcDataFmt >= 1 && nSelSrcDataFmt <= 7)
        {
            eSrcDataFmt =
                (nSelSrcDataFmt == 1) ? RGB16 :
                (nSelSrcDataFmt == 2) ? RGB24 :
                (nSelSrcDataFmt == 3) ? YCBYCR :
                (nSelSrcDataFmt == 4) ? YCRYCB :
                (nSelSrcDataFmt == 5) ? CBYCRY :
                (nSelSrcDataFmt == 6) ? CRYCBY : YC420;
            break;
        }
        else
            UART_Printf("Invalid Data Format! Retry It!!\n");
    }
#if 0 // jihyun
    while (true)
```

```
        {
            UART_Printf("\n");
            UART_Printf("[0] Exit\n");
            UART_Printf("[1] YUV444 (Y:10, CB:10 CR: 10) Interleave\n");
            UART_Printf("[2] RGB30 (R:10 G:10 B:10)\n");
            UART_Printf(">> Enter Destination Data Format: ");

            nSelDstDataFmt = UART_GetIntNum();

            if (nSelDstDataFmt == 0)
                return;
            else if (nSelDstDataFmt >= 1 && nSelDstDataFmt <= 2)
            {
                ePostFifoIf =       (nSelDstDataFmt == 1) ? YUV : RGB;
                break;
            }
            else
                UART_Printf("Invalid Data Format! Retry It!!\n");
        }
    #endif
        if (eSrcDataFmt == RGB16)
        {
            uImgHSz = 240, uImgVSz = 320;
            //H-Size : Scale down(280->240),      V-Size : Scale down(360->320)
            uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
            uMidScaledHSz = 280, uMidScaledVSz = 360;
        }
        else if (eSrcDataFmt == RGB24)
        {
```

```
        uImgHSz = 240, uImgVSz = 320;
        //H-Size : Scale up(200->240),      V-Size : Scale up(240->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
        uMidScaledHSz = 200, uMidScaledVSz = 240;
}
else if (eSrcDataFmt == YCBYCR)
{
        uImgHSz = 240, uImgVSz = 320;
        //H-Size : Scale up(200->240),      V-Size : Scale down(360->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 160, uMidStartY = 120;
        uMidScaledHSz = 200, uMidScaledVSz = 360;
}
else if (eSrcDataFmt == YCRYCB)
{
        uImgHSz = 240, uImgVSz = 320;
        //H-Size : Scale down(280->240),      V-Size : Scale up(280->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 0, uMidStartY = 0;
        uMidScaledHSz = 280, uMidScaledVSz = 280;
}
else if (eSrcDataFmt == CBYCRY)
{
        uImgHSz = 240, uImgVSz = 320;
        //H-Size : Scale up(200->240),      V-Size : Scale up(280->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
        uMidScaledHSz = 200, uMidScaledVSz = 280;
}
else if (eSrcDataFmt == CRYCBY)
{
        uImgHSz = 240, uImgVSz = 320;
```

```
        //H-Size : Scale down(320->240),      V-Size : Scale down(440->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 40;
        uMidScaledHSz = 320, uMidScaledVSz = 440;
    }
    else if (eSrcDataFmt == YC420)
    {
        uImgHSz = 240, uImgVSz = 320;
        //H-Size : Scale up(200->240),      V-Size : Scale down(400->320)
        uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 0;
        uMidScaledHSz = 320, uMidScaledVSz = 400;
    }
    UART_Printf("\n");
    if(uSimpleTest == 0)
    {
        UART_Printf("=========       Complex_Test Image Size       =========\n");
        UART_Printf("SrcImgHSz         = %d,             SrcImgVSz         = %d\n",uMidImgHSz,uMidImgVSz);
        UART_Printf("SrcStartX         = %d,             SrcStartY         = %d\n",uMidStartX,uMidStartY);
        UART_Printf("SrcCroppedHSz = %d,           SrcCroppedVSz = %d\n",uMidScaledHSz,uMidScaledVSz);
        UART_Printf("================================================\n");
        UART_Printf("\n");
    }
}
```

16.7 寄存器文件列表

下面是对各个寄存器的具体介绍。

16.7.1. 模式控制寄存器

寄存器	地址	读/写	描述	复位值
MODE	0X76300000	读/写	模式寄存器（[31:0]）。	0x00070B12

MODE	位	描述	初始状态
ExtFIFOIn	[31]	输入 FIFO 模式有效。1对应 FIFO 模式， 0 对应 DMA 模式。	0
CLKVALUP	[30]	CLKVAL_F 更新定时控制。 0 = 一直 1 = 一 帧的开始(一帧只有一次)	0
CLKVAL_F	[29:24]	确定TSCLK 和CLKVAL[5:0]的比率。 TSCLK = 时钟源 / (CLKVAL+1) 其中CLKVAL >= 1 注：VCLK 的最大频率是66MHz 。	0
CLKDIR	[23]	选择指定或者用CLKVAL_F 寄存器划分时钟源。 0 =指定的时钟(TSCLK的频率 = 时钟源频率) 1 = 用CLKVAL_F划分	0
CLKSEL_F	[22:21]	选择视频时钟源。 00 = HCLK 01 = PLL Ext 时钟输入 10 =保留 11 = 27MHz Ext 时钟输出	0
OutYCbCrFormat	[20:19]	当目标图像是交叉的YCbCr 格式，它决定字数据的字节组织。更多信息参考图6-12(b)。	0

OutRGB	[18]	它表示目标图像的输出色彩空间。0对应 YCbCr 或1 对应 RGB。	1
DST420	[17]	0 对应YCbCr422 ，1 对应 YCbCr420 目标格式。它只对于YCbCr目标图像是有效的 (也就是 OutRGB = 0)。	1
R2YSel	[16]	从RGB 到YCbCr选择色彩空间转变等式。1 对应 YCbCr 宽度范围， 0 对应 YCbCr 窄度范围。	1
InYCbCrFormat_ MSB	[15]	当源图像是交叉YCbCr格式，它决定字数据字节组织的MSB。	0
AutoLoadEnable	[14]	自动载入有效。 0 对应逐帧模式，1 对应自运行模式。	0
LCDPathEnable	[13]	输出FIFO模式有效 。 1 对应 FIFO 模式 ，0 对应DMA模式。	0
Interlace	[12]	只用在FIFO模式(LCDPathEnable =1)下的，输出浏览方法选择寄存器。 1对应隔行扫描，0 对应逐行扫描。 在 DMA 模式(LCDPathEnable = 0)下，不论给予何值都执行逐行扫描。	0
Wide/Narrow	[11:10]	根据输入值的范围选择从YCbCr 到RGB 色彩空间转变等式。 2’10 对应YCbCr 宽度范围，2’01 对应YCbCr 窄度范围。	2’b10
SRC420	[8]	0 对应YCbCr422 和1 对应 YCbCr420 源格式。它只有对于 YCbCr 源图像(也就是InRGB = 0) 是有效的。	1
INTEN	[7]	中断有效。当当前帧的处理完成时，由它决定 POSTINT 信号是否有效，。 0:无效 1:有效	0

POSTINT	[6]	中断等待位。如果INTEN有效，当当前帧完成操作，则自动声明正确。它必须被中断服务程序清除。 0: 无效， 1: 有效	0
IRQ_LEVEL	[5]	它决定电平产生方案。1对应电平中断(必须是1)。	0
OutRGBFormat	[4]	它决定目标图像的输出格式。 0对应16位(565格式) RGB 和1对应24位RGB。	1
InRGB	[3]	它指示源图像的输入色彩空间。 0 对应 YCbCr 或 1对应 RGB。	0
INTERLEAVE	[2]	它指示YCbC的数据格式。0 对应非交叉格式(Y、Cb和Cr分量以字存取)。 1 对应交叉格式(Y、 Cb 和 Cr 混合在单个字内)。当源图像是RGB 数据(或 InRGB =1)时，它应当是1。 。	0
InRGBFormat	[1]	如果源图像在RGB 色彩空间（或InRGB=1），它表示图解图像的数据格式。0对应16位 (565 格式) 和1 对应24位。 否则(或InRGB=0)，它应当保持为1。	1
InYCbCrFormat_LSB	[0]	当源图像是交叉YCbCr时，它决定字数据字节组织的LSB。	0

注意：[9]位为保留位

内部时钟设计模块图如图16-9所示。

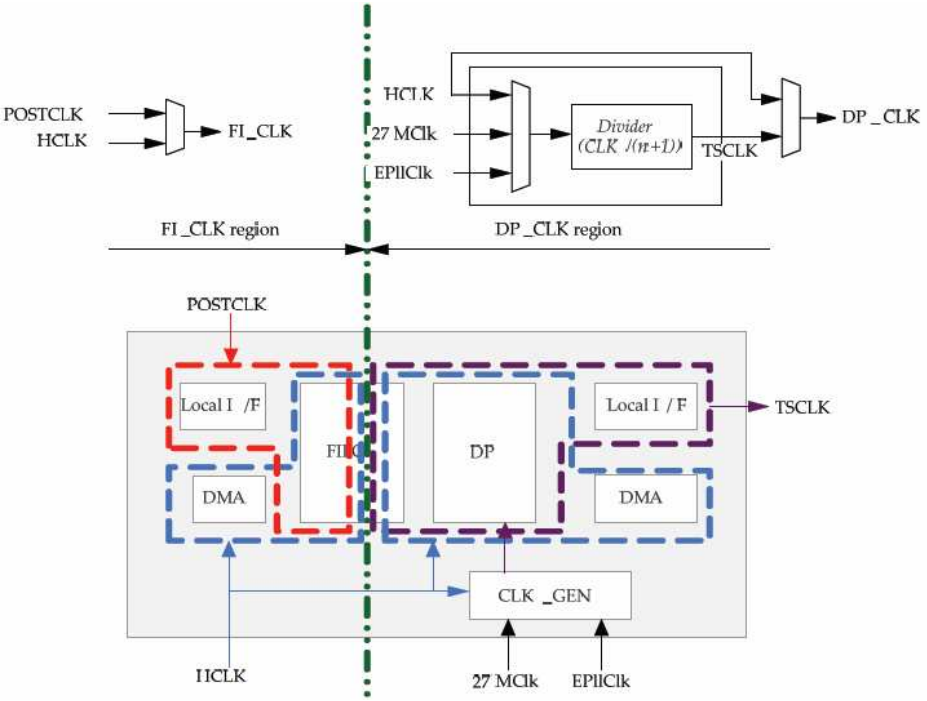

图16-9 内部时钟设计模块图

16.7.2. 预缩放比率寄存器

寄存器	地址	读/写	描述	复位值
PreScale_Ratio	0X76300004	读/写	用于垂直和水平方向预缩放比率。	0x0

PreScale_Ratio	位	描述	初始状态
PreScale_V_Ratio	[13:7]	预缩放比率，用于垂直方向。	0x0
PreScale_H_Ratio	[6:0]	预缩放比率，用于水平方向。	0x0

16.7.3. 预缩放图像大小寄存器

寄存器	地址	读/写	描述	复位值
PreScaleImgSize	0X76300008	读/写	预缩放图像大小。	0x0