Jacinto6 屏幕旋转显示解决方案

Fredy Zhang *EP FAE*

摘要

TI Jacinto6 家族的处理器包含两个系列，一个是专注于 ADAS 的 TDA2x/TDA3x 系列处理器，另外一个是专注于车载信息娱乐系统的 DRA7xx 处理器。Jacinto6 片上系统 (SoC) 是经过高度优化的可扩展系列器件，专为满足领先的高级驾驶辅助系统 (ADAS) 的要求而设计。TDA2xx 系列集最佳的性能、低功耗和 ADAS 视觉分析处理功能于一体，可广泛应 用于汽车领域中的 ADAS 应用，以推动实现更自主的无碰撞驾驶体验。DRA7xx 信息娱乐处理器系列的全面可扩展性，包括图像、语音、HMI、多媒体和智能手机投影模式功能，是专为信息娱乐应用设计的处理器，可支持视频图像和图形处理。

TI 提供了 HLOS PSDKLA（Linux）和 VISION SDK 两套软件用以支持相关的应用开发。 前者 PSDKLA 主要是运行在 A15 上的 Linux，后者 VISION SDK 是用以支持 TI ADAS 软件的框架。不论在 ADAS 应用中，还是在信息娱乐系统中，都需要支持图像处理及显示。车内的屏幕也多种多样，有横屏和有竖屏，再者摄像头的输入图像也有旋转显示的需求。因此，屏幕旋转显示方案对这样的应用至关重要。

本手册分别介绍了在 PSDKLA 和 VISION SDK 屏幕旋转显示解决方案，经验证，可用于量产屏幕旋转显示解决方案。

修改记录

Version	Date	Author	Notes
1.0	March 2022	Fredy Zhang	First release

目录

图

1. 基本介绍

TI TDA2x 和 TDA7xx 都是属于 TI Jacinto6 家族的汽车应用处理器。TDA2x 汽车应用处理器是经过高度优化的可扩展系列器件，专为满足领先的高级驾驶辅助系统 (ADAS) 的要求而设计。DRA7xx 汽车应用处理器旨在满足现代数字驾驶舱汽车体验需求而设计。

TI Jacinto6 家族的处理器，基于异构、可扩展的架构开发，此架构包含了 TI DSP 处理器、Cortex A15、Quad-Cortex M4、深度学习加速器 EVE、图形处理器 GPU 等核，属于多核异构的架构。Cortex A15 可用于通用计算、图形处理器 GPU 用于 3D 图像的加速、DSP 可用于算法的加速、EVE 可支持深度学习的处理、Cortex-M4 可用于外设的控制和图像的前后处理等。多核异构的优点是采用适合的核做擅长的事，再加上专用硬件加速器也可处理特定任务，从而在性能、功耗和成本 上达到最佳平衡。

如图 1 所示，Jacinto6 家族的有 TDA2x 和 TDA3x 处理器。 TDA3x 因没有配置 Cortex A15 和 GPU 核，因而不可以运行 Linux，适用于 2D SRV、CMS 等系统。TDA2x 处理其因配置了 Cortex A15 和 GPU 核可以运行高级操作系统（HLOS）Linux， 其处理器广泛应用于驾驶员监测、环视及泊车的应用。

图 **1**. Jacinto6 TDA2xx/TDA3xx Soc

如图 2 所示，Jacinto6 DRA7xx 处理器拥有多个系列的处理器（J6P、J6、J6Eco、J6 Entry），是面向数字仪表和车机市场的一个强大的平台，采用一颗 DRA7xx 处理器可实现的多种车机功能融合，包括传统的娱乐导航系统、数字仪表盘、抬头显示以及高级驾驶信息辅助系统，并可同时支持 Apple Carplay、Car Play 等手机互联功能。它通过可扩展的、成熟的、具有成本优势且软 件兼容的平台，以及成熟和全面的软件 SDK 和生态 系统的支持，满足了当前的数字仪表和车机解决方案要求。

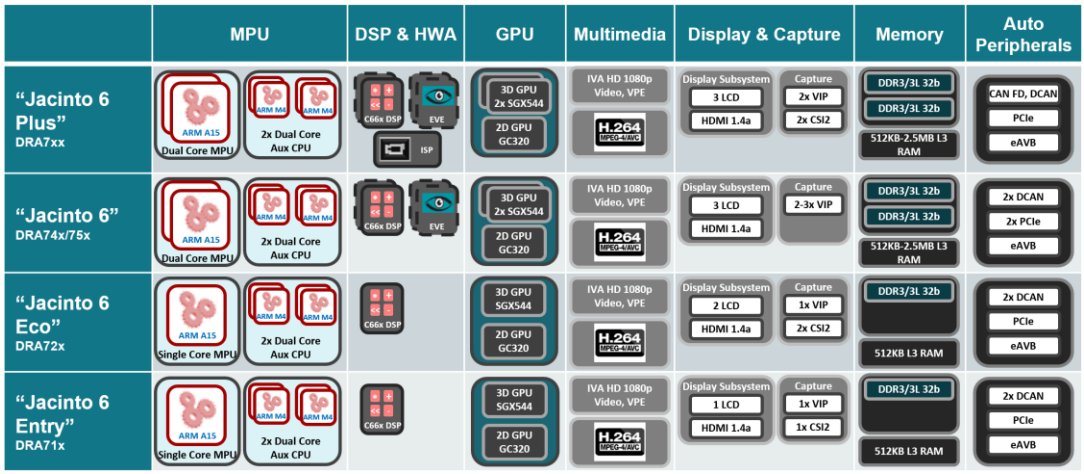

图 **2**. Jacinto6 DRA7xx Soc

TI TDA2xx 和 DRA7xx 处理器集成了 Imagination Tech SGX544 的 3D GPU，用于处理所有 3D 图形渲染以及支持 OpenGL ES 应用程序编程接口 (API)。不论是 ADAS 系统还是车机系统，都需要 UI 显示。

TI 在 Jacinto6 家族的处理器中提供了两套软件的 SDK，分别时 PSDKLA 和 VISION SDK。PSDKLA 是运行在 A72 上面的 HLOS Linux 操作系统， VISION SDK 是 TI ADAS 软件框架，用以支持 M4/C66x/EVE/A15 上面的 RTOS 软件。当前，在越来越多的 ADAS 应用场景中，比如泊车应用，都采用了 PSDKLA+VISION SDK 的软件架构，其软件架构如下：

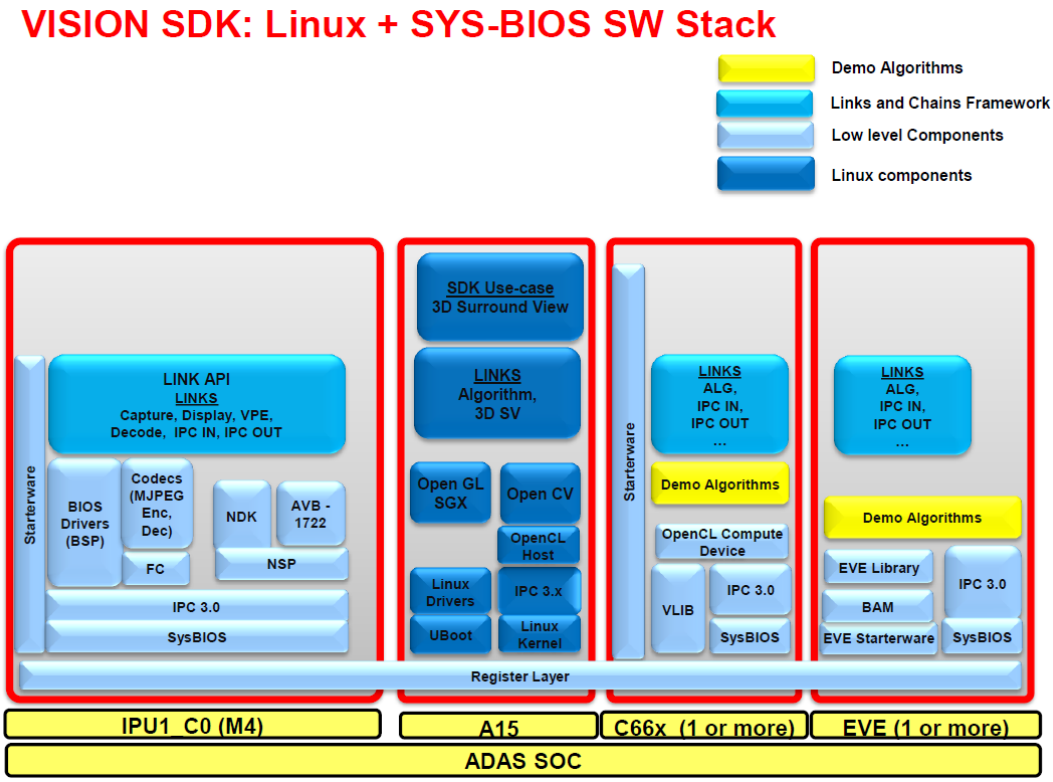

图 **3**. PSDKLA+VISION SDK SW Stack

当前手机、平板、电脑都支持旋转屏幕。如图 4 所示：屏幕有竖屏和横屏，图像的处理过程中，有些输入横的，有些输入时竖的。因此，在这些场景中，都有屏幕旋转显示的需求。因此，屏幕旋转显示方案对这些应用至关重要。 本文将分别介绍在 PSDKLA 和 VISION SDK 中屏幕旋转显示解决方案。

图 **4**. Screen Rotation

2. PSDKLA 屏幕旋转显示方案

TI DRA7xx 通常采用 PSDKLA SDK 进行软件开发。 TDA2xx 通常使用 VISION SDK, 大多数情况下会和 PSDKLA 一起使用。PSDKLA 是 A15 上的 Linux 开发环境，指的是 PROCESSOR-SDK-LINUX-AUTOMOTIVE。VISION SDK 是 TI ADAS 开发环境，主要支持 C6x DSP、EVE、M4 核上应用的开发，指的是 PROCESSOR-SDK-VISION。

Linux 开发环境中，通常使用 Weston 进行显示。Wayland/Weston 框架专用于显示器的管理，包括内容的合成，对其输入设备（触摸屏，鼠标，键盘）事件管理和其设备的配置（背景，壁纸，分辨率，多屏显示等）。Wayland 是一个指定显示器和客户端之间的通信协议。 Wayland 旨在替代 X Window 系统，更易于开发和维护。Weston 是 Wayland 合成器的参考实现，该合成器是使用 Wayland 协议的显示服务器，Weston 是小且快的合成器，适用于许多嵌入式和移动应用场景。

Weston 从内部体系结构如图 5 所示：主要分为窗口管理（Shell），合成器（compositor），渲染器（Renderer）和输入管理（input manager）几个部分。从流程上来讲，一方面输入模块接收用户输入，然后 Shell 做出相应的窗口管理操作；另一方面将该 input event 传给 Weston 之前注册了相应输入事件的 Client 程序. Client 收到后会在其处理函数中做相应动作，如调整视图重绘。如有重绘发生，client 端新 buffer 渲染完成后，client 将其句柄 handle 传给 server，之后是 compositor 合成后 renderer 进行渲染，最后输出到显示设备。Weston 启动过程中会分别加载几个模块：shell backend 用于窗口管理；render backend 用于合成渲染；compositor backend 用于合成输出；input manager backend 用于输入管理。

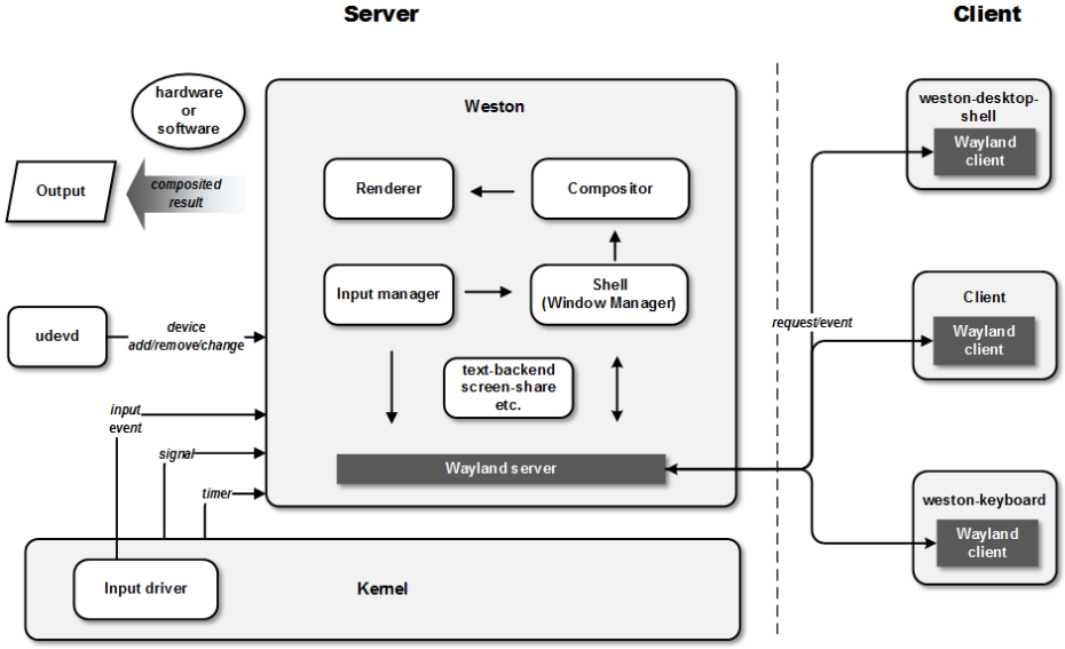

图 5. Weston 框架

Jacinto6 Linux 开发环境中，实现屏幕旋转的方案比较简单，通过对 weston 进行配置即可实现在 PSDKLA 中实现屏幕显示旋转。SDK 中通过查看 weston.ini 文件查看 weston 配置。

```
targetfs$ cat /etc/weston.ini
[shell]
locking=false
animation=zoom
panel-location=top
```

```
startup-animation=fade

[screensaver]
# Uncomment path to disable screensaver
#path=@libexecdir@/weston-screensaver
```

Jacinto6 PSDKLA 开发环境中，添加如下内容到 weston.ini 即可实现屏幕旋转。

```
[output]
name=Unknown-1   //LCD device name
transform=90     //Rotation 90 Degree Clockwise
```

3. VISION SDK 屏幕旋转显示方案

TI DRA7xx 通常采用 PSDKLA SDK 进行软件开发。 TDA2xx 通常使用 VISION SDK, 大多数情况下会和 PSDKLA 一起使用。PSDKLA 是 A15 上的 Linux 开发环境，指的是 PROCESSOR-SDK-LINUX-AUTOMOTIVE。VISION SDK 是 TI ADAS 开发环境，主要支持 C6x DSP、EVE、M4 核上应用的开发，指的是 PROCESSOR-SDK-VISION。 VISION SDK 当前已经包含了 PSDKLA 中的 Linux 开发环境，按照 SDK/vision_sdk/docs/Linux/VisionSDK_Linux_UserGuide.pdf 即可安装 A15 上运行的 Linux 环境。

值得说明的是在 PSDKLA+VISION SDK 的软件架构下，DSS 硬件被 M4（VISION SDK）所控制，Linux 不能直接控制 DSS（Display Subsystem）显示内容，Linux 需要通过 vDRM 显示内容。（对于 vDRM 可以参考 Integrating Virtual DRM Between VISION SDK and PSDK on Jacinto6 SOC）, DSS 在 M4 上的驱动并没有对屏幕旋转的实现，因此，如何在 VISION SDK 实现屏幕旋转显示方案是很多用户在实际应用过程中面临的重要问题。

从硬件机制上来说，Tiler 可以支持屏幕旋转，但是在 VISION SDK 没有相关实现，并且开发难度大。因此，如何利用现有的资源实现屏幕旋转至关重要。TI Soc 内部集成了 EDMA（Enhanced DMA），DMA（Direct Memory Access）可以实现数据的高效搬移和重新排列。本文提出了一种使用 EDMA 实现屏幕旋转的方案。接下来会详细介绍这种方案。

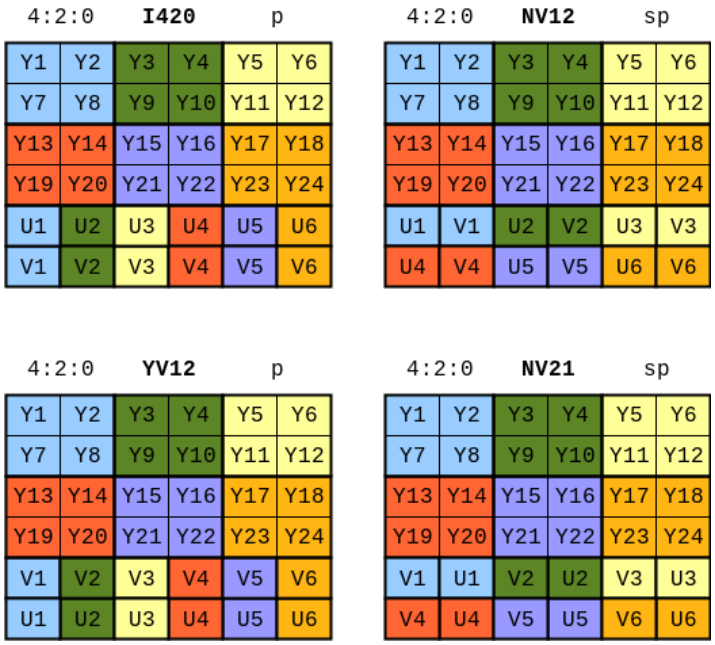

图 **6**. YUV420 格式说明

许多应用程序需要使用多个数据数组，任一数据通过外设传输数据的时候，数组一个接一个到达，每个数组占用一部分连续的内存空间。对于这些情况，EDMA 可以将数据重新组织到所需的格式。对于 YUV、RGB 等图像，在内存中也是以数组的方式的存在于连续的内存的空间。因此，利用 EDMA 硬件可以不经过 CPU 就可以将数据高效按规则搬移到指定内存空间。

在 VISION SDK 中，通常为了节约图像存储空间，我们使用 YUV420SP 格式存储在连续的内存的空间。如图 6 右 1 所示 YUV420SP 的图像格式，在实际的存储空间中，每帧图像 Y 存储在一段连续的内存空间中，UV 存储在另一段连续的内存的空间。相同的颜色的 Y 对应一组 UV。

我们先来找出图像搬移的规律，看一下怎么实现 YUV420 图像中 Y 旋转 90 度。在内存中实现 Y 旋转 90 度如下图 7 所示。EDMA 可以实现 3D 数据搬移，我们看一下怎么实现 Y 旋转 90 度。我们把每次从连续空间中搬移数据的大小用 ACNT，搬移的次数用 BCNT 表示，这样搬移数据的方式定义为一个 Chain，那么 CCNT 就可以表示为搬移一个数据块。使用 1280*720 分辨率的图像举例来说，那么实现 Y 数据块搬移的 EDMA 的配置如下：

- ACNT： 1 （每次从连续内存空间搬移数据的个数）
- BCNT： 1280 （连续搬移一行数据所需要的次数定义为一个 Chain）
- CCNT： 720 （分辨率的行数即需要多少个 Chain 才能实现整块 Y 数据搬移）
- SRCBIDX： 1 （源数据地址第一个数据到第二个数据的距离，即源数据地址 Y1 到 Y2 的距离）
- DSTBIDX： 720 （目的数据地址第一个数据到第二个数据的距离，即目的数据地址 Y1 到 Y2 的距离）
- DSTCIDX； -1 （目的数据地址第一行数据到第二行数据的距离，即 Y1 到 Y7 的距离，为负说明数据地址是递减的）

通过上述 EDMA 参数的配置，我们实现了逐个将数据 Y1、Y2… 搬移到了目的地址 Y19、Y13…。从而实现了图像旋转。

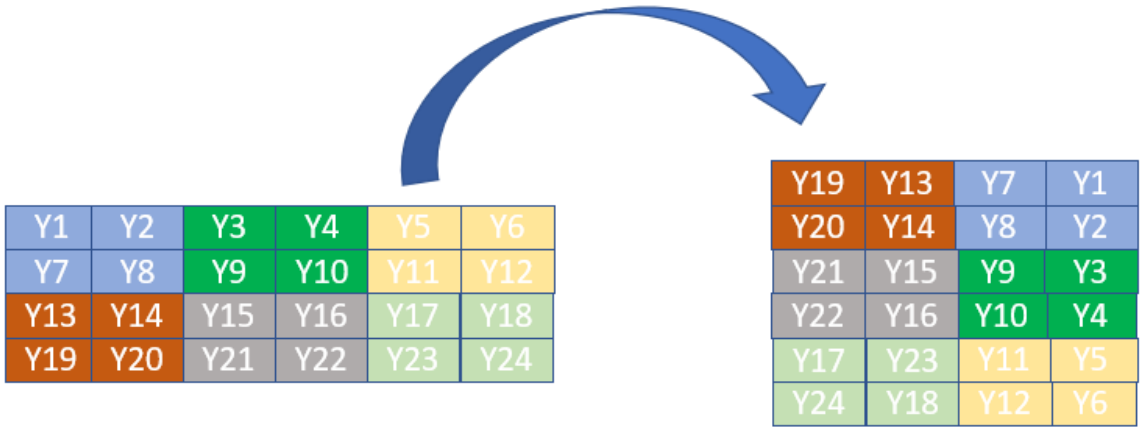

图 7. YUV420 Y 旋转 90 度

接下来，我们看一下怎么实现 YUV420 图像中 UV 顺时针旋转 90 度。在内存中实现 UV 旋转 90 度如下图 8 所示。使用 1280*720 分辨率的图像举例来说，那么实现 UV 数据块搬移的 EDMA 的配置如下：

- ACNT： 2 （每次从连续内存空间搬移数据的个数）
- BCNT：640 （连续搬移一行数据所需要的次数定义为一个 Chain）
- CCNT：360 （分辨率的行数即需要多少个 Chain 才能实现整块 Y 数据搬移）
- SRCBIDX： 2 （源数据地址第一个数据到第二个数据的距离，即源数据地址 U1 到 U2 的距离）
- DSTBIDX： 360 （目的数据地址第一个数据到第二个数据的距离，即目的数据地址 U1 到 U2 的距离）
- DSTCIDX； -2 （目的数据地址第一行数据到第二行数据的距离，即 U1 到 Y4 的距离）

通过上述 EDMA 参数的配置，我们把 UV 数据看作是一组数据。我们实现了逐个将数据 U1V1、U2V2… 搬移到了目的地址 U4V4、U1V1…。从而实现了图像旋转。

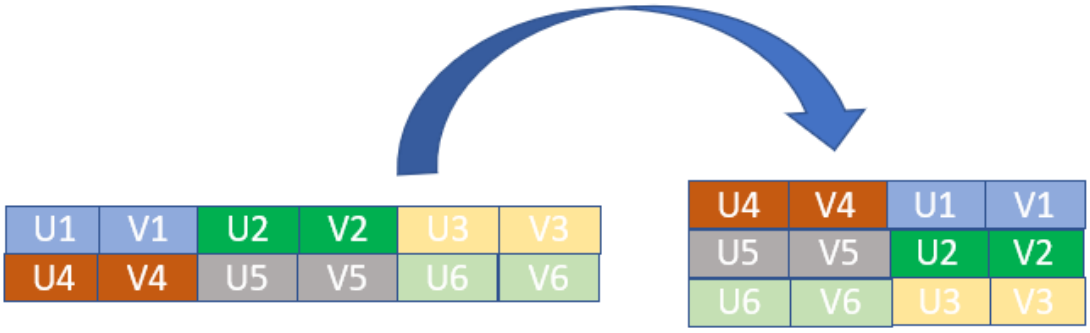

图 **8**. YUV420 UV 旋转 90 度

上面已经介绍了 YUV420SP 图像利用 EDMA 旋转的原理。在 VISION SDK 里面，我们具体要怎么来实现呢？VISION SDK 采用了 Links and Chains 软件框架，一个个数据处理的节点变成了一个个 LINK。 VISION SDK 已经支持 FrameCopy LINK 能够支持拷贝一帧当前的数据到目的 Buffer。因此，利用 FrameCopy Link，FrameCopy Link 支持 CPU 和 EDMA 拷贝，因此，我们可以利用 EDMA 拷贝的 LINK，更改 EDMA 配置快速实现基于 EDMA 的视频旋转方案。EDMA 主要配置内容 links_fw/src/rtos/utils_common/src/utils_dma.c 中 Utils_dmaCopyFill2D 函数，参考如下函数进行修改：

```
Int32 Utils_dmaCopyFill2D(Utils_DmaChObj  *pObj, const Utils_DmaCopyFill2D  *pInfo,
                UInt32 numTransfers, Bool fillData)
{
  EDMA3_DRV_Result  edma3Result = EDMA3_DRV_SOK;
  EDMA3_DRV_PaRAMRegs  *pParamSet;
  UInt32 bpp; /* bytes per pixel */
  UInt32 i, numTx;
  BspOsal_semWait(pObj->semLock,  BSP_OSAL_WAIT_FOREVER);
  numTx = 0U;
  for(i=0;  i<numTransfers;  i++){
    pParamSet   = pObj->txObj[numTx].pParamSet;
    if(pInfo->dataFormat==SYSTEM_DF_RAW24){
      bpp = 3U;
    }else if(pInfo->dataFormat==SYSTEM_DF_RAW16){
      bpp = 2U;
    }else if(pInfo->dataFormat==SYSTEM_DF_RAW08){
      bpp = 1U;
    }else if(pInfo->dataFormat==SYSTEM_DF_YUV420SP_UV){
      bpp = 1U;
    }else{
      bpp = 1U;
    }

    pParamSet->destAddr   = (UInt32)pInfo->destAddr[0]
                + (pInfo->destPitch[0]*pInfo->destStartY)
                + (pInfo->destStartX * bpp)+ ((pInfo->width)  -1);
```

```
if(fillData)
{
   pParamSet->srcAddr    = (UInt32)pInfo->srcAddr[0];
   /* MISRA.CAST.PTR_TO_INT
    * MISRAC_2004_Rule_11.3:Cast between a pointer and an integral type
    * State: Fix in later release -> Waiver -> Case by case
    * IP reads data in a particular format. So pointer to int conversion
    * is required to put data in that format.
    */
   pParamSet->srcBIdx    = 0;
} else{
   pParamSet->srcAddr    = (UInt32)pInfo->srcAddr[0]
                  + (pInfo->srcPitch[0]*pInfo->srcStartY)
                  + (pInfo->srcStartX * bpp);
   pParamSet->srcBIdx     = pInfo->srcPitch[0];
}
pParamSet->srcBIdx     = 1;
pParamSet->destBIdx    = pInfo->destPitch[0];

pParamSet->aCnt        = 1;
pParamSet->bCnt        = pInfo->height*bpp;
pParamSet->cCnt        = pInfo->width;

pParamSet->bCntReload  = pParamSet->bCnt;

pParamSet->srcCIdx     = pInfo->height*bpp;
                         pParamSet->destCIdx    = -1;
#ifdef SYSTEM_UTILS_DMA_PARAM_CHECK
UTILS_assert(Utils_dmaParamSetCheck(pParamSet)  == FVID2_SOK);
#endif

if(pInfo->dataFormat==SYSTEM_DF_YUV420SP_UV)
{
   numTx++;
   if(numTx>=pObj->maxTransfers){
      edma3Result += Utils_dmaRun(pObj, &numTx);
   }

   /* setup PaRAM for UV plane */
   pParamSet    = pObj->txObj[numTx].pParamSet;

   bpp = 1U;
   pParamSet->destAddr = (UInt32)pInfo->destAddr[1] + (pInfo->width) -2 ;
   if(fillData)
   {
      pParamSet->srcAddr    = (UInt32)pInfo->srcAddr[1];
```

```
            pParamSet->srcBIdx    = 0;
          }
          else
          {
            pParamSet->srcAddr = (UInt32)pInfo->srcAddr[1]
                        + (pInfo->srcPitch[1]
                        * (((pInfo->srcStartY+pInfo->height)/2U)-1U))
                        + (pInfo->srcStartX * bpp);
            pParamSet->srcBIdx    = -(Int32)1 * (Int32)pInfo->srcPitch[1];
          }

          pParamSet->srcAddr =  (UInt32)pInfo->srcAddr[1];
          pParamSet->srcBIdx    =  2;
          pParamSet->destBIdx   = pInfo->destPitch[1];

          pParamSet->aCnt       = 2;
          pParamSet->bCnt       = pInfo->height*bpp;
          pParamSet->cCnt       = (pInfo->width>>1);

          pParamSet->bCntReload  = pParamSet->bCnt;

          pParamSet->srcCIdx     = pInfo->height*bpp;
                                   pParamSet->destCIdx    = -2;

          #ifdef SYSTEM_UTILS_DMA_PARAM_CHECK
          UTILS_assert(Utils_dmaParamSetCheck(pParamSet)   == FVID2_SOK);
          #endif
        }
        /* goto next tranfer information */
        pInfo++;
        numTx++;

        if(numTx>=pObj->maxTransfers)
        {
          edma3Result += Utils_dmaRun(pObj, &numTx);
        }
      }

      if(numTx)  {
        edma3Result += Utils_dmaRun(pObj, &numTx);
      }

      BspOsal_semPost(pObj->semLock);
      return edma3Result;
    }
```

4. 总结

本文分别介绍了在 PSDKLA 和 VISION SDK 中屏幕旋转显示解决方案。基于 PSDKLA 屏幕旋转显示方案解决了在 A15 Linux 环境中的屏幕旋转显示问题。基于 EDMA 的图像旋转显示方案解决了在 VISION SDK 和 PSDKLA+VISION SDK 软件框架中图像旋转显示的需求。经验证，这两种方案都可以应用在量产解决方案中。解决了用户在实际使用中旋转显示的痛点问题。

5. 参考文献

1. https://www.ti.com.cn/cn/lit/wp/zhcy077/zhcy077.pdf
2. https://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Automotive_Software_Developers_Guide
3. https://www.ti.com/lit/ug/spru234c/spru234c.pdf
4. https://www.ti.com/lit/pdf/spracx5
5. https://www.ti.com/lit/pdf/sprui29

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265