AX VO 开发指南

文档版本：V2.1

发布日期：2024/09/02

目 录

权利声明

注意

您购买的产品、服务或特性等应受商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非商业合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

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。

前 言

本文档主要介绍如何在本 SDK 平台上进行 VO 相关的开发。

适用产品

AX620E 系列产品（AX630C、AX620Q）

适读人群

- 终端用户
- 售前
- 售后
- 技术人员

符号与格式定义

符号/格式	说明
`Xxx`	表示您可以执行的命令行。
斜体	表示变量。如，“*安装目录*/AX630C_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。
☞ 说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
！ 注意：	表示您在使用产品的过程中，需要您特别注意的事项。

修订历史

文档版本	发布时间	修订说明
V1.0	2024/01/30	文档初版
V1.1	2024/04/15	修改 2.2 屏相关配置 屏配置删除 display-timings 配置参数
V1.2	2024/06/20	补充 lvds 格式说明
V2.0	2024/07/17	增加图形层介绍章节 增加屏幕调试 FAQ 章节 修改屏幕调试介绍，补充各信号的测试示例
V2.1	2024/09/02	修改 4 开机 logo，开机 logo 支持 mipi 屏显示

1 概述

1.1 AxVO 简介

AxVO 是 AX620E 平台提供用于管理显示相关的模块。它基于 Linux Drm 框架实现的。

1.2 软件层次结构

图 1-1 所示是本芯片 VO 的软件层次结构，VO 驱动是基于 Linux DRM 框架针对本系列芯片 VO 硬件模块进行设计的。在用户空间 LIBAX_VO 基于 LIBDRM 进行封装向用户程序提供显示相关的接口。具体接口的使用请参考 AX VO API 文档。

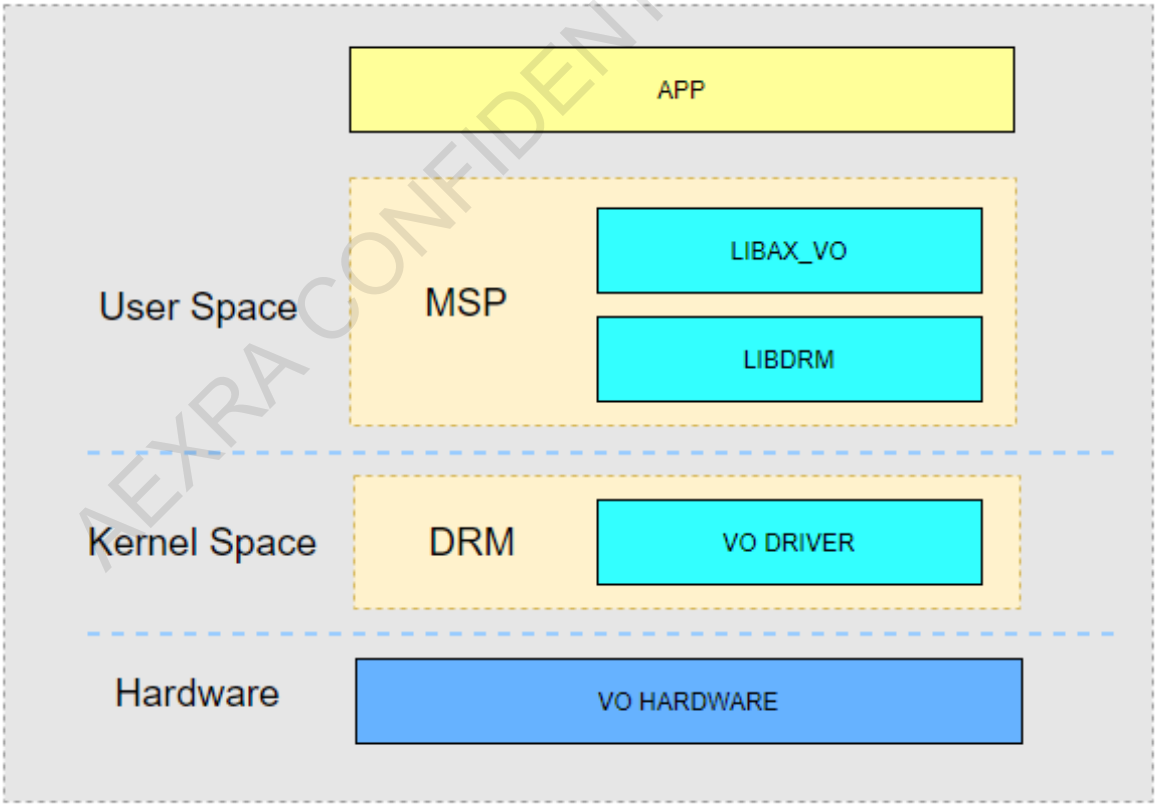

图1-1 AxVO 软件层次结构

1.3 显示信号

本芯片的 VO 单元集成了一个 Display Controller 和一个 Display Interface，它们和屏一起构成一个 Display Pipe。图 2-1 显示了 Display Pipe 的组成结构。

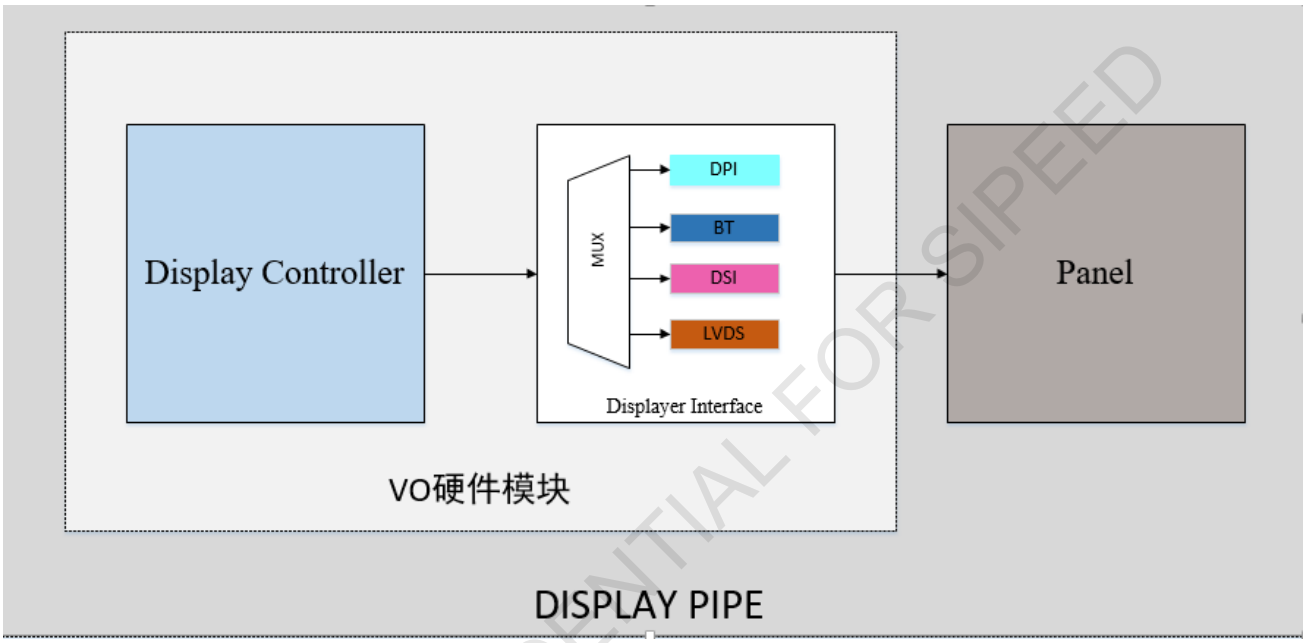

图1-2 Display Pipe 原理框图

Display Interface 支持 DPI、BT(BT565、BT601、BT1120)、MIPI-DSI、LVDS 四种接口标准。详见表 3-1 的接口信号支持情况。

2 图形层介绍

本芯片的显示模块支持图形界面开发，可以用 Linux Framebuffer 来完成 GUI 界面显示业务，同时为每个设备接口设计了提供独立的鼠标层。

图形层支持 colorkey、alpha 以及反色设置。鼠标光标层支持 alpha 和反色设置。

本芯片支持 2 路显示设备，同时支持 2 个图形层，图层和设备的关系如下：

表2-1 图层和设备节点对应关系

FB 设备文件	图形层	对应显示设备
/dev/fb0	G0	G0 在 dev0 设备上显示
/dev/fb1	G1	G1 在 dev1 设备上显示

如果使用光标鼠标业务，本芯片每个设备分别支持一路光标鼠标层，光标层和设备对应关系如下：

表2-2 鼠标层和设备节点对应关系

FB 设备文件	光标鼠标层	对应显示设备
/dev/fb2	Cursor0	Cursor0 在 dev0 设备上显示
/dev/fb3	Cursor1	Cursor1 在 dev1 设备上显示

用户使用图形层需要加载驱动并在 DTS 中进行参数配置。

2.2 模块加载

为了显示图形层，用户需要加载 ax_fb.ko 驱动。该驱动会根据 dts 中 vfb 节点的配置参数情况

创建 fb 设备文件，分配 FB 内存。图形层设备数据会和视频层数据做 blending 融合（blending 融合操作是在 ax_vo 驱动实现），如果使用了鼠标层，融合后的图象再叠加鼠标层，最后显示。

2.3 参数配置

ax_fb.ko 驱动加载、内存大小分配是由 dts 中的 vfb 节点情况来决定的。用户可根据自己的业务情况来做 dts 节点参数配置。

示例配置：

```
vfb0: vfb@0 {
        compatible = "axera,vfb";
        id = <0>;
        width = <1920>;
        height = <1080>;
        bpp = <32>;
        buf-num = <2>;
        status = "okay";
};

vfb1: vfb@1 {
        compatible = "axera,vfb";
        id = <1>;
        width = <1920>;
        height = <1080>;
        bpp = <32>;
        buf-num = <2>;
        status = "okay";
};

vfb2: vfb@2 {
        compatible = "axera,vfb";
        id = <2>;
        width = <64>;
        height = <64>;
        bpp = <32>;
        buf-num = <1>;
        cursor;
        status = "okay";
};
```

配置说明：该配置有三个 vfb 节点，驱动会创建三个 fb 文件。vfb0 和 vfb1 分别是 1920x1080 32bit 双 buffer，内存大小为 1920x1080x4x2，为图形层 fb。vfb2 为鼠标层 fb，大小为 64x64 32bit 单 buffer。

表2-3 Vfb 节点参数含义

属性	含义
compatible	固定为 axera,vfb.和 ax_fb 驱动中的字段相匹配，不可变更
width	图形层宽度
height	图形层高度
bpp	每个像素占用的位数；假设图形层数据格式为 ARGB8888，bpp 则为 32
buf-num	FB 缓存个数。1 为单个缓存，2 为双缓存
cursor	有此配置说明该节点为鼠标光标层，没有该项表示为图形层
status	节点状态。okay 表示开启，disabled 表示关闭

☞ **说明：**

- 用户可根据自己的需求配置，假设为单屏显示，有 GUI 和鼠标显示需求，则只需配置两个 vfb 节点：一个图形层+一个鼠标层即可。
- 建议用双缓存，即 buf-num 设置为 2，两个 buffe 交替读写，免出现显示画面撕裂闪烁等。
- fb 使用说明详见《AX_FB 开发指南》

3 LCD 屏调试

本系列芯片支持 DPI、BT(BT565、BT601、BT1120)、MIPI-DSI、LVDS 四种接口标准。但不同芯片存在差异。

表3-1 支持的接口类型

接口类型	AX630C		AX620Q	
	dev0	dev1	dev0	dev1
MIPI_DSI	支持	不支持	支持	不支持
LVDS_TX	支持	不支持	支持	不支持
DPI	支持	支持	不支持	不支持
BT	支持	支持	只支持 bt656	只支持 bt656

☞ **说明：**

- AX630C 芯片，DPI/BT 信号只能 dev0/dev1 二选一输出。
- AX620Q 芯片，因硬件管脚受限，只能支持 BT656 输出，BT656 和 MIPI 信号复用管脚，因此 MIPI/BT656 输出格式二选一。
- 若要双屏输出相同信号，只有 AX630C 能支持同时输出 BT656 格式，其中一个 bt656 输出走 MIPI 信号的通路。

3.1 DTS 节点介绍

VO 模块业务强依赖 DTS 配置，和 VO 模块相关的 DTS 节如下：

```
&drm0 {
        status = "okay";
};

&drm1 {
        status = "okay";
};

&crtc0 {
        status = "okay";
};

&crtc1 {
        status = "okay";
};

&vo0 {
        status = "okay";
};

&vo1 {
        status = "okay";
};

&bt_dpi0 {
        status = "okay";
};

&bt_dpi1 {
        status = "okay";
};

&dsi {
        status = "okay";
};

&panel_dsi {
        status = "okay";
};

&panel0 {
        status = "okay";
};

&panel1 {
        status = "okay";
};

&lvdstx {
        status = "okay";
};

&panel_lvds {
        status = "okay";
        data-mapping = "vesa-24";
};
```

表3-2 dts 显示配置节点说明

节点名称	节点说明
drm0	Drm 相关，控制 dpu0 的显示，用 dev0 显示时打开
drm1	Drm 相关，控制 dpu1 的显示，用 dev1 显示时打开
crtc0	Drm 相关，控制 dpu0 的显示，用 dev0 显示时打开
crtc1	Drm 相关，控制 dpu1 的显示，用 dev1 显示时打开
vo0	vo 驱动节点，加载 ax_vo 驱动需要打开
vo1	vo 驱动节点，使用 dev1 做显示业务需要打开
bt_dpi0	Drm 相关，dpu0 显示 BT/DPI 信号打开，和 panel0 节点开关保持一致
bt_dpi1	Drm 相关，dpu1 显示 BT/DPI 信号打开，和 panel1 节点开关保持一致 注意：dpu1 有两个通路 RGMII 和 MIPITX 可以出 BT656 信号 dmux-sel = <0>，表示从 MIPITX 管脚出 BT 信号，该通路只支持 BT656 dmux-sel = <1>，表示从 RGMII 管脚出 BT 信号
dsi	Drm 相关，dpu0 显示 MIPI 信号需要打开，和 panel_dsi 节点开关保持一致
lvdstx	Drm 相关，dpu0 显示 LVDS 信号需要打开，和 panel_lvds 节点开关保持一致
panel0	显示屏相关，dpu0 显示 BT/DPI 信号打开，和 bt-dpi0 节点开关保持一致
panel1	显示屏相关，dpu1 显示 BT/DPI 信号打开，和 bt-dpi1 节点开关保持一致

panel_dsi	显示屏相关，显示 MIPI 信号打开，和 dsi 节点开关保持一致
panel_lvds	显示屏相关，显示 LVDS 信号打开，和 lvdstx 节点开关保持一致

☞ **说明：**

用户可以默认所有相关节点都打开，如果做节点裁剪时需注意：

1. dev0 做输出时，需要打开 drm0、crtc0、vo0 节点以及输出信号节点

   dev1 做输出时，需要打开 drm1、crtc1、vo0、vo1 节点以及输出信号节点

2. 输出信号和对应屏节点要配套开关,对应关系：

   bt_dpi0-- panel0

   bt_dpi1-- panel1

   dsi -- panel_dsi

   lvds-- panel_lvds

3. bt 信号通路选择需配置 dts 属性

4. 时序参数不需要配置到 dts 中，用 API 接口设置

3.2 MIPI 屏调试

MIPI 输出只有 dev0 支持，dev1 不支持。

3.2.1 管脚配置

管脚默认为 MIPI 输出，一般情况下不需修改 pinmux。

3.2.2 DTS 配置

DTS 配置分为两部分：1）根据输出的物理设备接口选择打开对应的输出节点，dev0 做 MIPI 输出，需要保证 drm0、crtc0、vo0 节点开启；2）MIPI 输出信号节点开启。

MIPI 输出信号在 DTS 中需要为两个设备节点配置属性，分别为 dsi 节点和 panel_dsi 节点，

保证这两个节点为开启状态。panel_dsi 节点配置要求客户根据屏的需求填写合适的参数，节点参数含义如下：

表3-3 panel_dsi 节点参数含义

属性	含义
prepare-delay-ms	屏复位到下发初始化参数的延时
unprepare-delay-ms	可选，屏幕关闭后的延时
enable-delay-ms	可选，LP 切到 HS 模式的延时
disable-delay-ms	可选，屏幕退出工作模式的延时
reset-gpio	屏幕 GPIO 复位时 GPIO 配置
dsi,format	屏显示的数据格式，支持 RGB888\RGB666\RGB565 模式，具体参照 mipi_dsi_pixel_format 定义
dsi,lanes	lane 数目，支持 1-4
dsi,flags	burst/non-burst，时钟连续/非连续等参数配置 MIPI_DSI_MODE_VIDEO ：video 模式 MIPI_DSI_MODE_VIDEO_BURST：BURST MODE 传输数据 MIPI_DSI_MODE_VIDEO_SYNC_PULSE：NON BURST 同步脉冲模式传输数据 MIPI_DSI_CLOCK_NON_CONTINUOUS：非连续时钟 MIPI_DSI_MODE_LPM：DSI 低功耗
panel-init-seq	屏上电初始化序列，如果屏无需上电初始化序列建议发送一个空命令（05 00 01 00）。命令含义详见 5-3 章节
panel-exit-seq	屏的下电初始化序列，命令含义详见 5-3 章节
dsi-panel_esd_check_enable	静电检测相关配置；esd-check 配置使能：1 打开 0 关闭

dsi-panel-check-mode	静电检测相关配置；esd-check 方式，本芯片目前只支持读寄存器方式 0
dsi-panel-read-reg	静电检测相关配置；读寄存器方式监测的寄存器，取决于屏配置
dsi-panel-read-length	静电检测相关配置；dsi-panel-read-reg 寄存器要读寄存器的字节数，目前仅支持 1 个
dsi-panel-max-error-count	静电检测相关配置；寄存器值异常的容忍度，超过该次数会重配屏参做屏修复
dsi-panel-status-value	静电检测相关配置；dsi-panel-read-reg 寄存器的正确值
dsi-panel-check-interval-ms	静电检测相关配置；多长时间做一次 esd-check，时间单位 ms

3.2.3 时序配置

时序参数不需要在 DTS 中配置，通过 API 接口 AX_VO_SetPubAttr 配置时序，同时该接口也可配置输出信号类型，接口使用详细参考《24 - AX VO API 文档.docx》。

3.2.4 配置示例

屏幕情况：1080x1920@60 的 4lane video mode，像素格式为 RGB888。数据传输选用的时序模式为：Non-Burst Mode with Sync Pulses，时钟非连续。

屏幕需要复位，我们选用 GPIO_A0 来做屏幕复位。

1. pinmux 配置 MIPI 输出（一般情况下默认是 MIPI，不需要特殊处理）。
2. dts 配置，打开 dev0 相关节点：

```
&drm0 {
        status = "okay";
};

&crtc0 {
        status = "okay";
};

&vo0 {
        status = "okay";
};
```

3. dts 配置，打开 MIPI 信号节点，根据屏需求做参数配置：

```
&dsi {
        status = "okay";
};

&panel_dsi {
        status = "okay";
        prepare-delay-ms = <0>;
        unprepare-delay-ms = <0>;
        enable-delay-ms = <0>;
        disable-delay-ms = <150>;
        reset-gpio = <&ax_gpio1 0 0>;

        dsi,format = <MIPI_DSI_FMT_RGB888>;
        dsi,lanes = <4>;
        dsi,flags = <(MIPI_DSI_MODE_VIDEO_SYNC_PULSE | MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_LPM |MIPI_DSI_CLOCK_NON_CONTINUOUS )>;
        panel-init-seq = [39 00 04 B9 FF 83 99
                          39 00 10 B1 02 04 6B 8B 01 32 33 11 11 53 01 56 73 02 02
                          39 00 0c B2 00 80 80 AE 05 07 5A 11 00 10 10
                          39 00 2d B4 00 FF 00 AC 00 9C 00 00 08 00 00 02 00 26 02 07 09 21 03 00 00 04 A4 88 00 AC 00 9C 00 00 08 00 00 02 00 24 02 07 09 00 00 04 A4 12
                          39 05 22 D3 00 00 00 00 00 00 00 00 32 10 05 00 05 00 00 00 00 00 00 00 00 00 00 01 00 07 07 03 00 00 00 05 40
                          39 05 21 D5 00 00 00 00 01 02 03 00 00 00 00 19 00 18 00 20 21 00 18 00 00 00 00 00 00 00 00 31 31 30 30 2F 2F
                          39 05 21 D6 40 40 03 02 01 00 40 40 40 40 18 40 19 40 21 20 40 18 40 40 40 40 40 40 40 40 31 31 30 30 2F 2F
                          39 00 11 D8 28 02 00 2A 28 02 C0 2A 00 00 00 00 00 00 00 00
                          15 00 02 BD 01
                          39 00 11 D8 00 00 00 00 00 00 00 00 28 2A 00 2A 28 02 C0 2A
                          15 00 02 BD 02
                          39 00 09 D8 28 2A 00 2A 28 02 C0 2A
                          15 00 02 BD 00
                          39 00 03 B6 8D 8D
                          39 00 37 E0 01 49 4F 45 87 8A 8F 84 88 8D 91 94 97 9B A1 A2 A4 AA AB B0 A2 AF AF 59 54 5E 6D 01 49 4F 45 87 8A 8F 84 88 8D 91 94 97 9B A1 A2 A4 AA
                          15 00 02 D2 33
                          39 00 03 B6 7D 7D
                          39 00 08 BF 40 41 50 09 1A 80 CD
                          15 00 02 CC 08
                          15 05 02 36 00
                          05 00 01 35
                          05 00 01 29
                          05 05 01 11];
        panel-exit-seq = [05 00 01 10];
};
```

4. API 接口配置：

通过 AX_VO_SetPubAttr 配置 dev0 输出 DSI 信号，1080x1920@60 时序。

```
AX_S32 s32Ret = 0;
AX_VO_PUB_ATTR_T stVoPubAttr = {0};

stVoPubAttr.enMode = AX_VO_MODE_OFFLINE;
stVoPubAttr.enIntfType  = AX_VO_INTF_DSI;
stVoPubAttr.enIntfSync  = AX_VO_OUTPUT_1080x1920_60;
stVoPubAttr.enIntfFmt  = AX_VO_OUT_FMT_RGB888;

s32Ret = AX_VO_SetPubAttr(0, &stVoPubAttr);
if (s32Ret) {
    SAMPLE_PRT("failed with %#x, dev0\n", s32Ret);
    return s32Ret;
}
```

详细参照代码，case 运行指令：sample_vo –d 13

3.3 LVDS 屏调试

LVDS 输出只有 dev0 支持，dev1 不支持。

本芯片支持 6bit 和 8bit 的单路传输，不支持双路。

支持 VESA 和 JEIDA 两种标准。

3.3.1 管脚配置

一般情况下不需修改。

3.3.2 DTS 配置

DTS 配置分为两部分：1）根据输出的物理设备接口选择打开对应的输出节点，dev0 做 LVDS 输出，需要保证 drm0、crtc0、vo0 节点开启；2）LVDS 输出信号节点开启。

LVDS 输出信号在 DTS 中需要开启 lvdstx 节点，同时为 panel_lvds 节点配置属性且保证其为开启状态。panel_lvds 节点配置要求客户根据屏的需求填写合适的参数，节点参数含义如下：

表3-4 Panel-lvds 节点参数含义

属性	含义
status	节点使能/关闭
data-mapping	接口类型 可选配置： vesa-24: VESA 标准 单路 8bit jeida-24: JEIDA 标准 单路 8bit jeida-18：JEIDA 标准 单路 6bit

3.3.3 时序配置

时序参数不需要在 DTS 中配置，通过 API 接口 AX_VO_SetPubAttr 配置时序，同时该接口也可配置输出信号类型，接口使用详细参考《24 - AX VO API 文档.docx》。

3.3.4 配置示例

屏幕情况：1024x600 的 LVDS 屏幕，接口类型为单路 8bit，24 位 lvds 接口传输 rgb 数据。

屏幕规格书中表示支持 vesa 格式，如下：

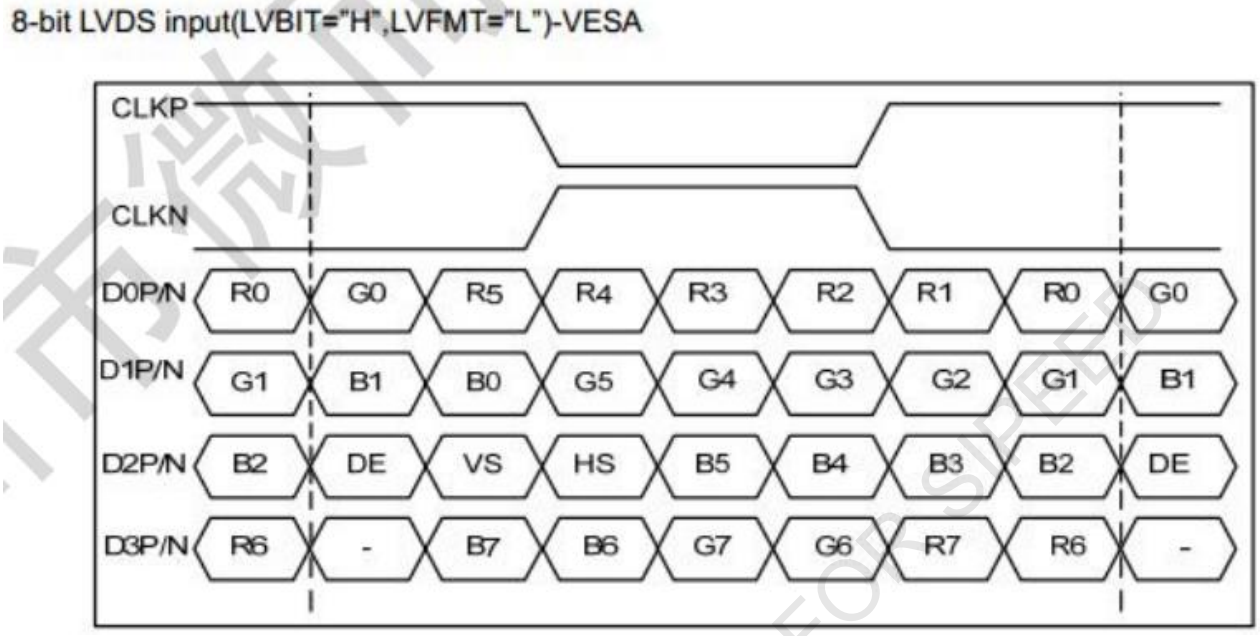

图3-1 示例屏支持的 8bit vesa 格式

根据上述信息，我们来配置 dev0 输出单路 8bit vesa 标准的 LVDS 信号，时序 1024x600@60。

1. pinmux 配置不需要特殊处理。

2. dts 配置，打开 dev0 相关节点：

```
&drm0 {
        status = "okay";
};

&crtc0 {
        status = "okay";
};

&vo0 {
        status = "okay";
};
```

3. dts 配置，打开 LVDS 信号节点，根据屏需求做参数配置：

```
&lvdstx {
        status = "okay";
};

&panel_lvds {
        status = "okay";
        data-mapping = "vesa-24";
};
```

4. API 接口配置：

通过 AX_VO_SetPubAttr 设置时序 1024x600@60，配置接口类型为 LVDS。

```
AX_S32 s32Ret = 0;
AX_VO_PUB_ATTR_T stVoPubAttr = {0};

stVoPubAttr.enMode = AX_VO_MODE_OFFLINE;
stVoPubAttr.enIntfType  = AX_VO_INTF_LVDS;
stVoPubAttr.enIntfSync  = AX_VO_OUTPUT_1024x600_60;
stVoPubAttr.enIntfFmt  = AX_VO_OUT_FMT_RGB888;

s32Ret = AX_VO_SetPubAttr(0, &stVoPubAttr);
if (s32Ret) {
    SAMPLE_PRT("failed with %#x, dev0\n", s32Ret);
    return s32Ret;
}
```

详细参照代码，case 运行指令：sample_vo –d 18

3.4 BT 信号调试

本系列芯片输出 BT 信号有两个硬件通络：一个是从 RGMII 侧 BT 信号，另一个是从 MIPITX 侧 BT 信号。因硬件管脚不同，不同芯片支持情况有差异：AX630C 芯片从 RGMII 侧 BT 信号可以支持 BT601\BT656\BT1120，MIPITX 侧 BT 只能支持 BT656。AX620Q 芯片因硬件管脚受限，只支持 MIPITX 侧出 BT656。

表3-5 BT 信号支持情况

输出信号	AX630C		AX620Q	
	DEV0	DEV1	DEV0	DEV1
RGMII 侧 BT 信号 (BT601/BT656/BT1120)	支持	支持	不支持	不支持
MIPITX 侧 BT656 信号	不支持	支持	不支持	支持

3.4.1 RGMII 侧出 BT 信号

只有 AX630C 支持，AX620Q 不支持，其他型号参考硬件手册。

3.4.1.1 管脚配置

和 RGMII 接口复用一组管脚，输出时 pinmux 修改为 BT 信号，详见 3.4.1.4 配置示例。

3.4.1.2 DTS 配置

DTS 配置分为两部分：根据输出的物理设备接口选择打开对应的输出节点，如果选择 dev0 做 BT 输出 1）需要保证 drm0、crtc0、vo0 节点开启；2）BT 输出信号 bt_dpi0 和 panel0 开启。

3.4.1.3 时序配置

时序参数不需要在 DTS 中配置，通过 API 接口 AX_VO_SetPubAttr 配置时序，同时该接口也可配置输出信号类型以及像素格式，接口使用详细参考《24 - AX VO API 文档.docx》。

3.4.1.4 配置示例

假设 AX630C 芯片要从 dev0 输出 720x480@60 的 BT601 信号，逐行扫描格式。配置如下：

1. pinmux 配置

管脚配置为 BT 信号，修改方法：将\build\tools\pinmux\AX630C_DEMO_pinmux.xlsm 下面 20 个 pinmux 配置为 DPI 信号。

27		UART3_RXD		0x90								1	SPI_M2_MISO	1	0	0
28		EMAC_PTP_PPS0		0xC								1	DPI_D17	1	0	0
29		EMAC_PTP_PPS1		0x18								1	DPI_D16	1	0	0
30		EMAC_PTP_PPS2		0x24								1	DPI_D18	0	0	0
31		EMAC_PTP_PPS3		0x30								1	DPI_D15	1	0	1
32		RGMII_MDCK		0x3C								1	DPI_D1	1	0	0
33		RGMII_MDIO		0x48								1	DPI_D2	1	0	0
34		EPHY_CLK		0x54								1	DPI_PCLK	0	0	0
35		EPHY_RSTN		0x60								1	DPI_D0	0	0	0
36		EPHY_LED0		0x6C								0	EPHY_LED0	1	0	0
37		EPHY_LED1		0x78								0	EPHY_LED1	1	0	0
38	G8	RGMII_RXD0	0x104F0000	0x84								1	DPI_D5	0	0	0
39		RGMII_RXD1		0x90								1	DPI_D6	0	0	0
40		RGMII_RXDV		0x9C								1	DPI_D4	0	0	0
41		RGMII_RXCLK		0xA8								1	DPI_D3	0	0	0
42		RGMII_RXD2		0xB4								1	DPI_D7	0	0	0
43		RGMII_RXD3		0xC0								1	DPI_D8	0	0	0
44		RGMII_TXD0		0xCC								1	DPI_D11	0	0	0
45		RGMII_TXD1		0xD8								1	DPI_D12	0	0	0
46		RGMII_TXCLK		0xE4								1	DPI_D9	0	0	0
47		RGMII_TXEN		0xF0								1	DPI_D10	0	0	0
48		RGMII_TXD2		0xFC								1	DPI_D13	0	0	0
49		RGMII_TXD3		0x108								1	DPI_D14	0	0	0
50		SD_DAT0		0xC								0	SD_DAT0	1	0	0
51		SD_DAT1		0x18								0	SD_DAT1	1	0	0

图3-2 BT 信号 RGMII 接口 pinmux 修改

export 后同级目录下生成 pinmux_AX630C_DEMO_pinmux.h，修改合入 build\projects 对应工程下的 pinmux\AX630C_DEMO_pinmux.h（只修改 20 个管脚）

```
0x104F000C, 0x00010083,  /* PadName = EMAC_PTP_PPS0        Fuction = DPI_D17 */
0x104F0018, 0x00010083,  /* PadName = EMAC_PTP_PPS1        Fuction = DPI_D16 */
0x104F0024, 0x00010003,  /* PadName = EMAC_PTP_PPS2        Fuction = DPI_D18 */
0x104F0030, 0x00010093,  /* PadName = EMAC_PTP_PPS3        Fuction = DPI_D15 */
0x104F003C, 0x00010083,  /* PadName = RGMII_MDCK           Fuction = DPI_D1 */
0x104F0048, 0x00010083,  /* PadName = RGMII_MDIO           Fuction = DPI_D2 */
0x104F0054, 0x00010003,  /* PadName = EPHY_CLK             Fuction = DPI_PCLK */
0x104F0060, 0x00010003,  /* PadName = EPHY_RSTN            Fuction = DPI_D0 */
0x104F006C, 0x00000083,  /* PadName = EPHY_LED0            Fuction = EPHY_LED0 */
0x104F0078, 0x00000083,  /* PadName = EPHY_LED1            Fuction = EPHY_LED1 */
0x104F0084, 0x00010005,  /* PadName = RGMII_RXD0           Fuction = DPI_D5 */
0x104F0090, 0x00010005,  /* PadName = RGMII_RXD1           Fuction = DPI_D6 */
0x104F009C, 0x00010003,  /* PadName = RGMII_RXDV           Fuction = DPI_D4 */
0x104F00A8, 0x00010005,  /* PadName = RGMII_RXCLK          Fuction = DPI_D3 */
0x104F00B4, 0x00010005,  /* PadName = RGMII_RXD2           Fuction = DPI_D7 */
0x104F00C0, 0x00010005,  /* PadName = RGMII_RXD3           Fuction = DPI_D8 */
0x104F00CC, 0x00010005,  /* PadName = RGMII_TXD0           Fuction = DPI_D11 */
0x104F00D8, 0x00010005,  /* PadName = RGMII_TXD1           Fuction = DPI_D12 */
0x104F00E4, 0x00010005,  /* PadName = RGMII_TXCLK          Fuction = DPI_D9 */
0x104F00F0, 0x00010003,  /* PadName = RGMII_TXEN           Fuction = DPI_D10 */
0x104F00FC, 0x00010005,  /* PadName = RGMII_TXD2           Fuction = DPI_D13 */
0x104F0108, 0x00010005,  /* PadName = RGMII_TXD3           Fuction = DPI_D14 */
```

2. 修改 dts 配置打开相关节点

```
&drm0 {
        status = "okay";
};

&crtc0 {
        status = "okay";
};

&vo0 {
        status = "okay";
};

&bt_dpi0 {
        status = "okay";
};

&panel0 {
        status = "okay";
};
```

3. API 接口配置

通过 AX_VO_SetPubAttr 配置 dev0 输出 BT601 信号，720x480@60 时序。

```
AX_S32 s32Ret = 0;
AX_VO_PUB_ATTR_T stVoPubAttr = {0};

stVoPubAttr.enMode = AX_VO_MODE_OFFLINE;
stVoPubAttr.enIntfType  = AX_VO_INTF_BT601;
stVoPubAttr.enIntfSync  = AX_VO_OUTPUT_480P60;
stVoPubAttr.enIntfFmt   = AX_VO_OUT_FMT_YUV422;

s32Ret = AX_VO_SetPubAttr(0, &stVoPubAttr);
if (s32Ret) {
    SAMPLE_PRT("failed with %#x, dev%d\n", s32Ret, pstVoDevConf->u32VoDev);
    return s32Ret;
}
```

详细参照代码，case 运行指令：sample_vo –d 17

3.4.2 MIPITX 侧出 BT 信号

只有 dev1 能从 MIPITX 侧输出 BT 信号，且只支持 BT656，不支持 BT601 和 BT1120。因管脚复用，dev0 输出 MIPI 信号和 dev1 用此通路输出 BT656 信号二选一使用。

3.4.1.5 管脚配置

和 MIPITX 接口复用一组管脚，输出时 pinmux 修改为 BT 信号，详见 3.4.2.4 配置示例。

3.4.1.6 DTS 配置

DTS 配置分为两部分：根据输出的物理设备接口选择打开对应的输出节点，如果选择 dev1 从 MIPITX 管脚做 BT 输出 1）需要保证 drm1、crtc1、vo0、vo1 节点开启；2）BT 输出信号 bt_dpi1 和 panel1 开启。

因 dev1 内部最多有两个硬件通路可以输出 BT 信号，所以要进行通路选择，配置如下：

```
&bt_dpi1 {
        status = "okay";
        dmux-sel = <0>;
};
```

☞ 说明：

dmux-sel = <0>，表示从 MIPITX 管脚出 BT 信号,该通路只支持 BT656

dmux-sel = <1>，表示从 RGMII 管脚出 BT 信号

3.4.1.7 时序配置

时序参数不需要在 DTS 中配置，通过 API 接口 AX_VO_SetPubAttr 配置时序，同时该接口也可配置输出信号类型以及像素格式，接口使用详细参考《24 - AX VO API 文档.docx》。

3.4.1.8 配置示例

以 AX620Q DEMO 板为例，要从 MIPITX 侧输出 720x480@60 的 BT656 信号，隔行扫描格式。配置如下：

1. pinmux 配置

管脚配置为 BT 信号，修改方法：将\build\tools\pinmux\AX620Q_DEMO_pinmux.xlsm 下面 9 个 pinmux 配置为 BT 信号。

dphytx	SLEEP_CTRL	0x230A000	0x4									
	CDTX_L0N		0xC								1	BT656_CLK
	CDTX_L0P		0x18								1	BT656_D0
	CDTX_L1N		0x24								1	BT656_D1
	CDTX_L1P		0x30								1	BT656_D2
	CDTX_L2N		0x3C								1	BT656_D3
	CDTX_L2P		0x48								1	BT656_D4
	CDTX_L3N		0x54								1	BT656_D5
	CDTX_L3P		0x60								1	BT656_D6
	CDTX_L4N		0x6C								1	BT656_D7
	CDTX_L4P		0x78								0	CDTX_L4P

图3-3 BT 信号 mipitx 管脚 pinmux 修改

export 后同级目录下生成 pinmux_AX620Q_DEMO_pinmux.h，修改合入 build\projects 对应工程下的 pinmux\AX620Q_DEMO_pinmux.h（只修改 9 个管脚）。

```
0x0230A00C, 0x00010003,  /* PadName = CDTX_L0N          Fuction = BT656_CLK */
0x0230A018, 0x00010003,  /* PadName = CDTX_L0P          Fuction = BT656_D0 */
0x0230A024, 0x00010003,  /* PadName = CDTX_L1N          Fuction = BT656_D1 */
0x0230A030, 0x00010003,  /* PadName = CDTX_L1P          Fuction = BT656_D2 */
0x0230A03C, 0x00010003,  /* PadName = CDTX_L2N          Fuction = BT656_D3 */
0x0230A048, 0x00010003,  /* PadName = CDTX_L2P          Fuction = BT656_D4 */
0x0230A054, 0x00010003,  /* PadName = CDTX_L3N          Fuction = BT656_D5 */
0x0230A060, 0x00010003,  /* PadName = CDTX_L3P          Fuction = BT656_D6 */
0x0230A06C, 0x00010003,  /* PadName = CDTX_L4N          Fuction = BT656_D7 */
```

2. 修改 dts 配置打开相关节点，选择 MIPITX 侧输出 BT656 信号通路。

```
&drm1 {
        status = "okay";
};

&crtc1 {
        status = "okay";
};

&vo0 {
        status = "okay";
};

&vo1 {
        status = "okay";
};

&bt_dpi1 {
        status = "okay";
        dmux-sel = <0>;
};

&panel1 {
        status = "okay";
};
```

3. API 接口配置

通过 AX_VO_SetPubAttr 配置 dev1 输出 BT656 信号，720x480@60 时序。注意 dev 只能是 dev1。

```
AX_S32 s32Ret = 0;
AX_VO_PUB_ATTR_T stVoPubAttr = {0};

stVoPubAttr.enMode = AX_VO_MODE_OFFLINE;
stVoPubAttr.enIntfType  = AX_VO_INTF_BT656;
stVoPubAttr.enIntfSync  = AX_VO_OUTPUT_480I60;
stVoPubAttr.enIntfFmt   = AX_VO_OUT_FMT_YUV422;

s32Ret = AX_VO_SetPubAttr(1, &stVoPubAttr);
if (s32Ret) {
    SAMPLE_PRT("failed with %#x, dev0\n", s32Ret);
    return s32Ret;
}
```

3.4.3 BT 外接转换芯片

如果外接 BT 转换芯片，可将外接芯片挂在 I2C 上，客户自己添加对应的驱动即可。

下面举个 AX620Q DEMO 板外接 BT 转 CVBS 信号的例子，出 BT 的配置参照 3.4.1 和 3.4.2，这里只介绍外接芯片的调试

1）假设外设挂在 I2C1 上，对应的 dts 中在 I2C1 节点下加入外设子节点：

```
&i2c1 {
        #address-cells = <0x1>;
        #size-cells = <0x0>;
        status = "okay";
        es7243l: es7243l@16 {
                compatible = "es7243l_MicArray_0";
                reg = <0x16>;
                #sound-dai-cells = <0>;
        };
        es8156: es8156@9 {
                compatible = "everest,es8156";
                reg = <0x9>;
                #sound-dai-cells = <0>;
        };
        ax_gpio_ext: gpio-axera@20 {
                compatible = "axera,gpio-ext";
                reg = <0x20>;
                gpio-controller;
                #gpio-cells = <2>;
                nr-gpios = <16>;
                status = "okay";
        };
        tp2803: tp2803@45 {
                status = "okay";
                compatible = "axera,tp2803";
                reg = <0x45>;
                reset-gpio = <&ax_gpio1 0 0>;
                panel-format = <PANEL_TP2803_BT656_720x576_PAL>;
        };
};
```


图3-4 I2C 挂接转换芯片

☞ **说明：**

reg 为 I2C 访问的外设（转接芯片）的寄存器地址。

reset-gpio 转接芯片复位，这里选择的是 GPIO_A0 复位，可根据实际使用情况修改。

2）自行添加转接芯片相关驱动，可参照 drivers/gpu/drm/panel/panel-tp2803.c

3.5 DPI 信号调试

本系列芯片 AX630C 支持输出 DPI 信号，AX620Q 芯片因硬件管脚受限不支持。

DPI 信号输出只支持 RGB565。

表3-6 DPI 信号支持情况

输出信号	AX630C		AX620Q	
	DEV0	DEV1	DEV0	DEV1
DPI 信号	支持	支持	不支持	不支持

3.5.1 管脚配置

和 RGMII 接口复用一组管脚，输出时 pinmux 修改为 DPI 信号（和 BT 一样的修改），详见图 3-5。

3.5.2 DTS 配置

DTS 配置和从 RGMII 侧出 BT 信号的配置相同，两者共用一组节点。DTS 配置分为两部分：根据输出的物理设备接口选择打开对应的输出节点，如果选择 dev0 做 DPI 输出 1）需要保证 drm0、crtc0、vo0 节点开启；2）DPI 输出信号 bt_dpi0 和 panel0 开启。

3.5.3 时序配置

时序参数不需要在 DTS 中配置，通过 API 接口 AX_VO_SetPubAttr 配置时序，同时该接口也可配置输出信号类型以及像素格式，接口使用详细参考《24 - AX VO API 文档.docx》。

3.5.4 配置示例

以 AX630C 芯片要从 dev0 输出 800x480@60 的 DPI 信号。配置如下：

1. pinmux 配置

管脚配置为 DPI 信号，修改方法：将\build\tools\pinmux\AX630C_DEMO_pinmux.xlsm 下面 20 个 pinmux 配置为 DPI 信号。

27		UART3_RXD		0x90									1	SPI_M2_MISO	1	0	0		
28		EMAC_PTP_PPS0		0xC									1	DPI_D17	1	0	0		
29		EMAC_PTP_PPS1		0x18									1	DPI_D16	1	0	0		
30		EMAC_PTP_PPS2		0x24									1	DPI_D18	0	0	0		
31		EMAC_PTP_PPS3		0x30									1	DPI_D15	1	0	1		
32		RGMII_MDCK		0x3C									1	DPI_D1	1	0	0		
33		RGMII_MDIO		0x48									1	DPI_D2	1	0	0		
34		EPHY_CLK		0x54									1	DPI_PCLK	0	0	0		
35		EPHY_RSTN		0x60									1	DPI_D0	0	0	0		
36		EPHY_LED0		0x6C									0	EPHY_LED0	1	0	0		
37		EPHY_LED1		0x78									0	EPHY_LED1	1	0	0		
38	G8	RGMII_RXD0	0x104F0000	0x84									1	DPI_D5	0	0	0		
39		RGMII_RXD1		0x90									1	DPI_D6	0	0	0		
40		RGMII_RXDV		0x9C									1	DPI_D4	0	0	0		
41		RGMII_RXCLK		0xA8									1	DPI_D3	0	0	0		
42		RGMII_RXD2		0xB4									1	DPI_D7	0	0	0		
43		RGMII_RXD3		0xC0									1	DPI_D8	0	0	0		
44		RGMII_TXD0		0xCC									1	DPI_D11	0	0	0		
45		RGMII_TXD1		0xD8									1	DPI_D12	0	0	0		
46		RGMII_TXCLK		0xE4									1	DPI_D9	0	0	0		
47		RGMII_TXEN		0xF0									1	DPI_D10	0	0	0		
48		RGMII_TXD2		0xFC									1	DPI_D13	0	0	0		
49		RGMII_TXD3		0x108									1	DPI_D14	0	0	0		
50		SD_DAT0		0xC									0	SD_DAT0	1	0	0		
51		SD_DAT1		0x18									0	SD_DAT1	1	0	0		

图3-5 DPI 信号 pinmux 修改

export 后同级目录下生成 pinmux_AX630C_DEMO_pinmux.h，修改合入 build\projects 对应工程下的 pinmux\AX630C_DEMO_pinmux.h（只修改 20 个管脚）。

```
0x104F000C, 0x00010083,  /* PadName = EMAC_PTP_PPS0          Fuction = DPI_D17 */
0x104F0018, 0x00010083,  /* PadName = EMAC_PTP_PPS1          Fuction = DPI_D16 */
0x104F0024, 0x00010003,  /* PadName = EMAC_PTP_PPS2          Fuction = DPI_D18 */
0x104F0030, 0x00010093,  /* PadName = EMAC_PTP_PPS3          Fuction = DPI_D15 */
0x104F003C, 0x00010083,  /* PadName = RGMII_MDCK             Fuction = DPI_D1 */
0x104F0048, 0x00010083,  /* PadName = RGMII_MDIO             Fuction = DPI_D2 */
0x104F0054, 0x00010003,  /* PadName = EPHY_CLK               Fuction = DPI_PCLK */
0x104F0060, 0x00010003,  /* PadName = EPHY_RSTN              Fuction = DPI_D0 */
0x104F006C, 0x00000083,  /* PadName = EPHY_LED0              Fuction = EPHY_LED0 */
0x104F0078, 0x00000083,  /* PadName = EPHY_LED1              Fuction = EPHY_LED1 */
0x104F0084, 0x00010005,  /* PadName = RGMII_RXD0             Fuction = DPI_D5 */
0x104F0090, 0x00010005,  /* PadName = RGMII_RXD1             Fuction = DPI_D6 */
0x104F009C, 0x00010003,  /* PadName = RGMII_RXDV             Fuction = DPI_D4 */
0x104F00A8, 0x00010005,  /* PadName = RGMII_RXCLK            Fuction = DPI_D3 */
0x104F00B4, 0x00010005,  /* PadName = RGMII_RXD2             Fuction = DPI_D7 */
0x104F00C0, 0x00010005,  /* PadName = RGMII_RXD3             Fuction = DPI_D8 */
0x104F00CC, 0x00010005,  /* PadName = RGMII_TXD0             Fuction = DPI_D11 */
0x104F00D8, 0x00010005,  /* PadName = RGMII_TXD1             Fuction = DPI_D12 */
0x104F00E4, 0x00010005,  /* PadName = RGMII_TXCLK            Fuction = DPI_D9 */
0x104F00F0, 0x00010003,  /* PadName = RGMII_TXEN             Fuction = DPI_D10 */
0x104F00FC, 0x00010005,  /* PadName = RGMII_TXD2             Fuction = DPI_D13 */
0x104F0108, 0x00010005,  /* PadName = RGMII_TXD3             Fuction = DPI_D14 */
```

2. 修改 dts 配置打开相关节点

```
&drm0 {
        status = "okay";
};

&crtc0 {
        status = "okay";
};

&vo0 {
        status = "okay";
};

&bt_dpi0 {
        status = "okay";
};

&panel0 {
        status = "okay";
};
```

3. API 接口配置

通过 AX_VO_SetPubAttr 配置 dev0 输出 DPI 信号，800x480@60 时序。

```
AX_S32 s32Ret = 0;
AX_VO_PUB_ATTR_T stVoPubAttr = {0};

stVoPubAttr.enMode = AX_VO_MODE_OFFLINE;
stVoPubAttr.enIntfType  = AX_VO_INTF_DPI;
stVoPubAttr.enIntfSync  = AX_VO_OUTPUT_800x480_60;
stVoPubAttr.enIntfFmt  = AX_VO_OUT_FMT_RGB565;

s32Ret = AX_VO_SetPubAttr(0, &stVoPubAttr);
if (s32Ret) {      sunjing, 18 months ago • [MC20E-168] ISP: Sam
    SAMPLE_PRT("failed with %#x, dev0\n", s32Ret);
    return s32Ret;
}
```

详细参照代码，case 运行指令：sample_vo –d 0

4 开机 logo

本芯片目前支持 DPI 和 MIPI 信号的开机 LOGO，默认支持 DPI。

本芯片支持 bmp 和 jpg 两种图片格式。

4.1 Config 文件的配置

在 boot/uboot/u-boot-2020.04/configs 路径下对应的 defconfig 文件，使能相关配置，如图：

```
### below for boot logo###
CONFIG_DM_VIDEO=y
CONFIG_DISPLAY=y
CONFIG_VIDEO_AXERA=y
CONFIG_VIDEO_AXERA_MAX_XRES=2048
CONFIG_VIDEO_AXERA_MAX_YRES=2048
CONFIG_VIDEO_AXERA_DISPLAY_WIDTH=800
CONFIG_VIDEO_AXERA_DISPLAY_HEIGHT=480
CONFIG_DISPLAY_AXERA_DPI=y
# CONFIG_DISPLAY_AXERA_MIPI is not set
### above for boot logo ###
```

图4-1 使能 BOOT LOGO 相关的 defconfig 配置

相关说明：

CONFIG_VIDEO_AXERA：控制在 uboot 阶段是否做 logo 显示，配置为 y 表示开启；无此需求可不配置。

CONFIG_VIDEO_AXERA_MAX_XRES/ CONFIG_VIDEO_AXERA_MAX_YRES：logo 图片为 jpg 格式时需要设置，用来分配缓存，一般不低于图片的宽高；bmp 图片格式可以不用设置。

CONFIG_VIDEO_AXERA_DISPLAY_WIDTH / CONFIG_VIDEO_AXERA_DISPLAY_HEIGHT：logo 图片为 jpg 格式时需要设置，一般为显示屏对应像素的宽高。bmp 图片格式可以不用设置。

当 jpg 图片的宽高等于 CONFIG_VIDEO_AXERA_DISPLAY_xx 宽高时，内部会默认优先用 jpeg 硬解。

CONFIG_DISPLAY_AXERA_DPI：显示信号为 DPI 时配为 y。

CONFIG_DISPLAY_AXERA_MIPI：显示信号为 mipi dsi 时配为 y。

以上配置项也可以通过 menuconfig 设置。

4.2 Board 头文件的配置

在 boot/uboot/u-boot-2020.04/include/configs 路径下对应的 board 头文件中有如下配置：

```
/* boot-logo related buffer address */
#define LOGO_IMAGE_LOAD_ADDR 0x70000000U
#define LOGO_SHOW_BUFFER 0x71000000U
#define LOGO_SHOW_BUF_SIZE 0x800000U
```

图4-2 使能 BOOT LOGO 需要在 BOARD 头文件中增加的配置

相关说明：

LOGO_SHOW_BUFFER 用于指定 vo 的显示地址，此地址配置需谨慎，要避开 kernel/dtb 的加载地址，另外此地址在系统开机后还会继续使用。

LOGO_IMAGE_LOAD_ADDR 用于指定 logo 中的 RGB 或者 YUV 数据的存放地址。

在 boot/uboot/u-boot-2020.04/drivers/video/axera/ax_simple_logo.h 头文件中修改 AX_MIN_RESIZE_WIDTH 和 AX_MIN_RESIZE_HEIGHT 定义为实际屏的宽高。

4.3 屏时序相关配置

屏时序配置在/boot/uboot/u-boot-2020.04/drivers/video/axera/ax_vo.c 中，时序配置如下，新增时序需要配置到此：

```
struct display_timing fixed_timigs[AX_VO_OUTPUT_BUTT] = {
    { /* AX_VO_OUTPUT_1080P60 */
        .pixelclock = {0, 148500000, 0},
        .hactive = {0, 1920, 0},
        .hfront_porch = {0, 88, 0},
        .hback_porch = {0, 148, 0},
        .hsync_len = {0, 44, 0},
        .vactive = {0, 1080, 0},
        .vfront_porch = {0, 4, 0},
        .vback_porch = {0, 36, 0},
        .vsync_len = {0, 5, 0},
        .flags = 0xa,
    },
    { /* AX_VO_OUTPUT_800_480_60 */
        .pixelclock = {0, 29232000, 0},
        .hactive = {0, 800, 0},
        .hfront_porch = {0, 40, 0},
        .hback_porch = {0, 40, 0},
        .hsync_len = {0, 48, 0},
        .vactive = {0, 480, 0},
        .vfront_porch = {0, 13, 0},
        .vback_porch = {0, 29, 0},
        .vsync_len = {0, 3, 0},
        .flags = 0x15,
    },
};
```


图4-3 UBOOT-LOGO 时序配置

4.4 MIPI 屏初始化配置

显示屏为 mipi 屏时还需配置/boot/uboot/u-boot-2020.04/drivers/video/axera/ax_simple_logo_cfg.c 文件中的参数：

```
struct mipi_dsi_panel_cfg g_dsi_panel = {
        .nlanes = 4,
        .format = MIPI_DSI_FMT_RGB888,
        .mode_flags = MIPI_DSI_MODE_VIDEO_SYNC_PULSE | MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_LPM |MIPI_DSI_CLOCK_NON_CONTINUOUS,
        .gpio_num = 32,
        .panel_init_seq = init_seq,
        .init_seq_len = sizeof(init_seq)/sizeof(init_seq[0]),
        .reset_delay_ms = 100,
};
```

其中参数和 kernel 中 dts 中 panel-dsi 中的参数一致，可参照表 3.3。

屏幕初始化参数写入全局数组 `init_seq` 中，格式要求同 `kernel`，参照 5.3 章节。

`gpio_num`:屏幕选用 `GPIO` 做复位时，该参数为对应 `GPIO` 的 `num`。

4.5 LOGO 文件的支持

目前本芯片上 logo 文件支持 bmp 和 JPG 格式的文件，相关限制：

1. Bmp 文件的图像格式必须是 rgb565 或 rgb888，也就是 bmp 文件的 header 结构中 bit_counts 必须是 16 或者 24.
2. Logo 的宽(width)与位深(bpp)的乘积必须满足 16 对齐，即 width * bit_counts 必须是 16 对齐的，图像的高(height)必须 2 对齐.
3. 由于 vo 硬件相关的限制要求 bmp 的 data offset 必须 16 对齐，通常的 bmp 的 data offset 是 0x36，因此为了支持 vo 的硬件需要对 bmp 文件做些处理，目前 SDK 供了一个小工具来完成这个处理。

   工具路径：tools/boot_logo_tool

   方法：./ logo_proc -i axera_logo_orign.bmp -o axera_logo.bmp

   将上面生成的 axera_logo.bmp 存放到 boot/uboot/u-boot-2020.04/tools/logos/目录下，编译打包后 logo 文件会一起打包进去。
4. 本芯片支持 JPG 格式图片，实际的 JPG 图片宽高和显示宽高一致时默认硬解，否则软解。满足硬解条件时注意确认 JPEG_DECODE_WIDTH 和 JPEG_DECODE_HEIGHT 宏定义，需要和屏的宽高一致。

5 屏幕调试 FAQ

5.1 用户自定义时序怎么配

SDK 中提供了一组枚举定义了一些常见的标准时序，详见 header/external/ax_vo_api.h 中 AX_VO_INTF_SYNC_E。用户在调屏时可能需要使用自定义时序模式。

调用 AX_VO_SetPubAttr 接口设置属性时，接口时序选择为 AX_VO_OUTPUT_USER，按照屏的时序要求配置时序结构体 stSyncInfo，具体实例可参考 sample_vo –d 26。

```
typedef struct axVO_SYNC_INFO_T {
    AX_BOOL bSynm;      /* sync mode(0:timing,as BT.656; 1:signal,as LCD) */
    AX_BOOL bIop;       /* interlaced or progressive display(0:i; 1:p) */
    AX_U16 u16Vact;     /* vertical active area */
    AX_U16 u16Vbb;      /* vertical back blank porch */
    AX_U16 u16Vfb;      /* vertical front blank porch */
    AX_U16 u16Hact;     /* herizontal active area */
    AX_U16 u16Hbb;      /* herizontal back blank porch */
    AX_U16 u16Hfb;      /* herizontal front blank porch */
    AX_U16 u16Hmid;     /* bottom herizontal active area */
    AX_U16 u16Bvact;    /* bottom vertical active area */
    AX_U16 u16Bvbb;     /* bottom vertical back blank porch */
    AX_U16 u16Bvfb;     /* bottom vertical front blank porch */
    AX_U16 u16Hpw;      /* horizontal pulse width */
    AX_U16 u16Vpw;      /* vertical pulse width */
    AX_U32 u32Pclk;     /* pixel clock, in kHz */
    AX_BOOL bIdv;       /* inverse data valid of output */
    AX_BOOL bIhs;       /* inverse horizontal synch signal */
    AX_BOOL bIvs;       /* inverse vertical synch signal */
} AX_VO_SYNC_INFO_T;
```

表5-1 自定义时序结构体参数说明

参数	说明
bSynm	同步模式，本芯片无效参数

bIop	0：隔行　1：逐行
u16Vact	垂直有效区。单位：行
u16Vbb	垂直消隐后肩。单位：行
u16Vfb	垂直消隐前肩。单位：行
u16Hact	水平有效区。单位：像素
u16Hbb	水平消隐后肩。单位：像素
u16Hfb	水平消隐前肩。单位：像素
u16Hmid	本芯片无效参数
u16Bvact	本芯片无效参数
u16Bvbb	本芯片无效参数
u16Bvfb	本芯片无效参数
u16Hpw	水平同步信号的宽度。单位：像素。
u16Vpw	垂直同步信号的宽度。单位：像素。
u32Pclk	像素时钟
bIdv	数据有效信号的极性
bIhs	水平有效信号的极性
bIvs	垂直有效信号的极性

像素时钟与垂直/水平信号及刷新率之间的关系：

hblank = u16Hfb + u16Hpw + u16Hbb

htotal = hblank + u16Hact

vblank = u16Vfb + u16Vpw + u16Vbb

vtotal = vblank + u16Bvact

u32Pclk = (vtotal * htotal)* vrefresh

以 800x480@60 输出为例：

htotal = 900，vtotal = 500，vrefresh=60，Pixel Clock = 27000000，因此可算出：

u32Pclk = (900 * 500) *60 = 27000000

5.2 MIPI 屏怎么支持静电检测

ESD（Electro-Static discharge）的意思是“静电释放”，为了防止静电对电子器件造成的损害，需要做静电消除。本芯片支持软件 ESD CHECK，通过软件监控屏幕状态来检测和响应静电放电，避免因静电原因造成的显示异常。异常后做屏复位操作。

软件上通过读寄存器方式做 ESD，只需在 DTS 中对 panel_dsi 节点静电检测参数做相关配置即可，配置要求由屏决定，参数的具体含义请参考表 3-3。

假设某款屏可以通过监测 DISPLAY POWER MODE（寄存器 0x0A）监查屏否在正常工作，该屏手册显示显示 0x0A 寄存器回读只有一个参数，正常工作时该值为 0x9D，2s 检测一次，连续 3 次检测错误则判定为屏异常。则 panel_dsi 节点中配置参数如下：

```
dsi-panel_esd_check_enable = /bits/ 8 <1>;
dsi-panel-check-mode = /bits/ 8 <0>;
dsi-panel-read-reg = /bits/ 8 <0x0A>;
dsi-panel-read-length = /bits/ 8 <1>;
dsi-panel-max-error-count = /bits/ 8 <3>;
dsi-panel-status-value = /bits/ 8 <0x9D>;
dsi-panel-check-interval-ms = <2000>;
```

5.3 MIPI 屏初始化参数怎么转换为 DTS 参数格式

panel_dsi 节点 panel-init-seq 参数表示的是屏初始化参数，含义如下：

```
panel-init-seq = [39 00 06 FF 77 01 00 00 13
                  15 00 02 EF 08
                  39 00 06 FF 77 01 00 00 10
                  39 00 03 C0 2C 00
                  39 00 03 C1 08 02
                  39 00 03 C2 37 05
                  15 00 02 C3 02
                  15 00 02 CC 10
                  39 00 11 B0 04 0E 17 0B 0F 06 08 08 08 24 04 11 0F 2C 33 13
                  39 00 11 B1 0C 16 1D 0E 11 06 08 08 08 24 05 13 11 2D 33 1F
                  39 00 06 FF 77 01 00 00 11
                  ...
                  05 78 01 11
                  15 00 02 36 08
                  15 00 02 35 00
                  05 14 01 29];
```

图5-1 DSI 初始化命令

格式说明：头部 3 个字节（16 进制），分别代表 Data Type，Delay，Payload Length。从第四个字节开始的数据代表长度为 Length 的 Payload。

第一条命令的解析如下：

```
39 00 06 FF 77 01 00 00 13
```

Data Type: 0x39 (DCS Long Write)

Delay: 0x00 (0 ms)

Payload Length: 0x6 (6 Bytes)

Payload: 0xFF 0x77 0x01 0x00 0x00 0x13

最后一条命令的解析如下：

```
05 14 01 29
```

Data Type: 0x05 (DCS Short Write, no parameters)

Delay: 0x14 (20 ms)

Payload Length: 0x1 (1 Bytes)

Payload: 0x29

Data Type 和 DCS 中的 DATA TYPE 含义一致，如下：

表5-2 Data Type 含义说明

Data Type	描述	类型
0x05	DCS Short Write,no parameters	Short
0x15	DCS Short Write,1 parameters	Short
0x39	DCS Long Write/write_LUT Command Packet	Long
0x03	Generic Short Write,no parameters	Short
0x13	Generic Short Write,1 parameters	Short
0x23	Generic Short Write,2 parameters	Short
0x29	Generic Long Write	Long

DCS Write

DCS packet 包括一个字节的 dcs 命令，以及 n 个字节的 parameters。如果 n < 2，将以 Short Packet 的形式对 Payload 进行打包。n = 0，表示只发送 dcs 命令，不带参数，Data Type 为 0x05；n = 1，表示发送 dcs 命令，带一个参数，Data Type 为 0x15。如果 n ≥ 2，将以 Long Packet 的形式对 Payload 进行打包。此时发送 dcs 命令，带 n 个参数，Data Type 为 0x39。

Generic Write

Gerneic Packet 包括 n 个字节的 parameters。如果 n < 3， 将以 Short Packet 的形式对 Payload 进行打包。 n = 0， 表示 no parameters，Data Type 为 0x03； n = 1，表示 1 parameter， Data Type 为 0x13； n = 2， 表示 2 parameters， Data Type 为 0x23。如果

n >= 3，将以 Long Packet 的形式进行对 Payload 打包，表示 n parameters， Data Type 为 0x29。

Delay

表示当前 Packet 发送完成之后，需要延时多少 ms，再开始发送下一条命令。

Payload Length

表示 Packet 的有效负载长度。

Payload

表示 Packet 的有效负载，长度为 Payload Length。

5.4 图像偏移怎么调整

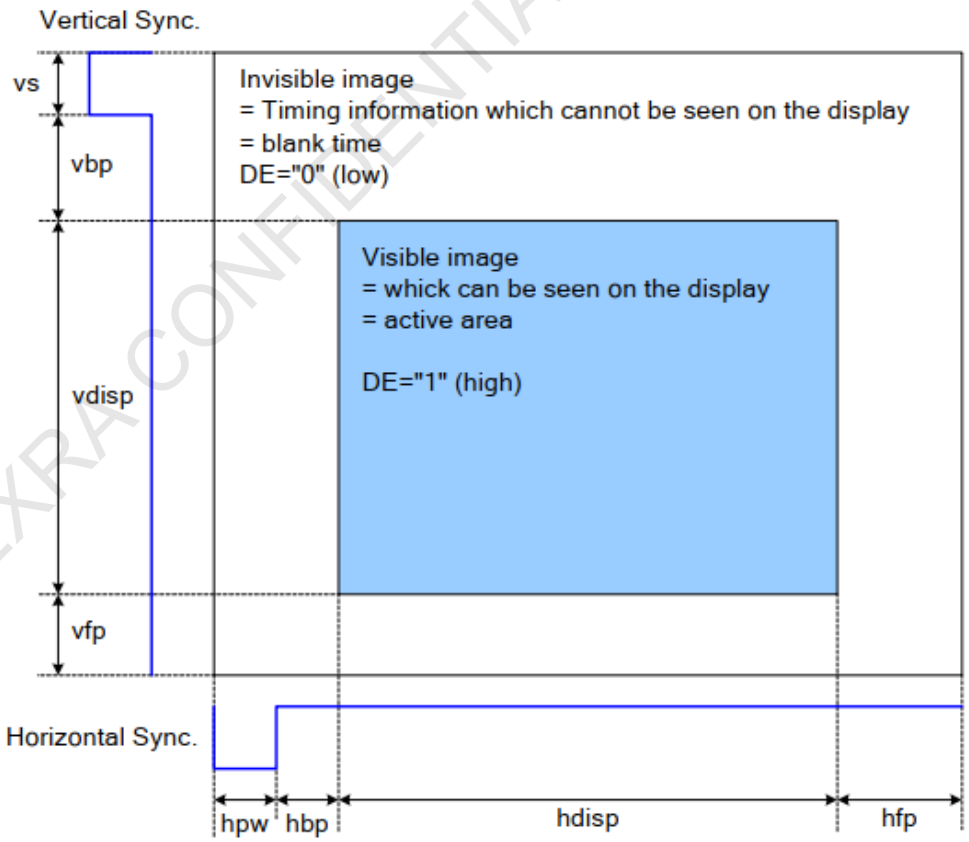

图5-2 控制信号和时序参数

当出现屏显图像整体向一个方向偏移，叠不叠加 GUI 一样的效果，且 DPU 处理图像无异常时。可以考虑根据实际偏移方向按照图 5-2 调整时序参数中的水平消隐前肩/后肩，垂直消隐前肩/后肩这些参数来移动图像。假设显示图像整体偏左，则可以调大 hbp，缩小 hfb。