Rockchip RK3588 MIPI-DSI2 软件开发指南

文件标识：RK-KF-YF-705

发布版本：V2.0.0

日期：2022-07-06

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

文本主要对 RK3588 MIPI DSI2 各种场景应用进行软件配置说明。

产品版本

芯片名称	内核版本
RK3588	LINUX Kernel 5.10

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

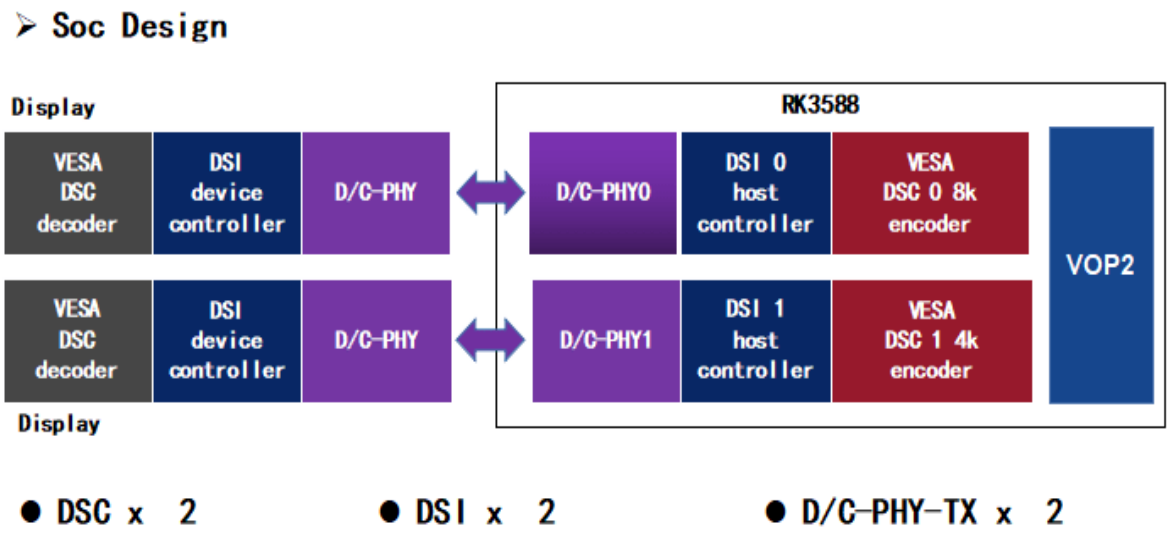
修订记录

版本号	作者	修改日期	修改说明
V1.0.0	黄国椿	2022-01-07	初始发布
V2.0.0	黄国椿	2022-07-06	补充DSC等内容

目录

Introduction

DSI-2 是 MIPI 联盟定义的一组通信协议的一部分， DWC-MIPI-DSI2 是一个实现 MIPI-DSI2 规范中定义的所有协议功能的数字核控制器，可以兼容 D-PHY 和 C-PHY 的物理接口，支持两路的 Display Stream Compression (DSC) 数据传输, RK3588 有两个 DSI-2 控制器和两个独立的物理的 D/C-PHY, 可以同时最多支持两路 MIPI 输出。



MIPI-DSI2 Features

1. MIPI® Alliance Specification for Display Serial Interface 2 (DSI-2) Version 1.1
2. MIPI® Alliance Specification for Display Command Set (DCS) Version 1.4
3. MIPI® Alliance Specification for D-PHY v2.0
4. MIPI® Alliance Specification for C-PHY v1.1
5. Four data lanes on D-PHY and three data trios on C-PHY
6. Bidirectional communication and escape mode through data lane 0
7. Continuous and non-continuous clock modes on D-PHY and non-continuous clock mode on C-PHY
8. End of Transmission packet (EoTp)
9. Scrambling
10. VESA DSC 1.1/1.2a
11. Up to 4.5 Gbps per lane in D-PHY
12. Up to 2.0 Gsps per trio in C-PHY

MIPI DSI-2 Host 与 MIPI DSI Host 的差别

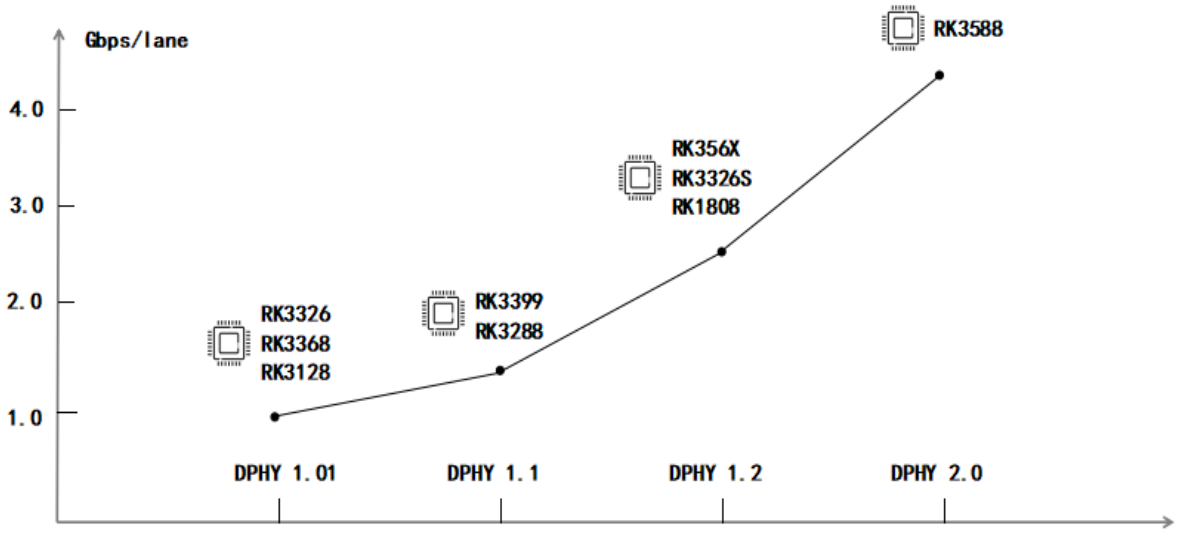
MIPI DSI-2 除了可以兼容 MIPI DSI 的所有协议功能外, 还增加支持 MIPI C-PHY.

	D-PHY 1.01	D-PHY 1.1	D-PHY 1.2	D-PHY 2.0	C-PHY
DSI 1.0	yes	yes	yes	yes	no
DSI 1.1	yes	yes	yes	yes	no
DSI 1.2	yes	yes	yes	yes	no
DSI 1.3	yes	yes	yes	yes	no
DSI-2 1.0	yes	yes	yes	yes	yes

MIPI DSI 兼容性

MIPI DPHY 差别

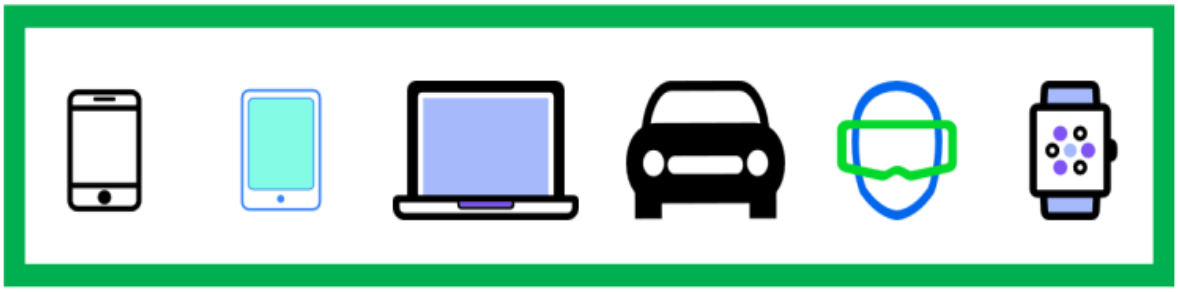
RK3588 平台 MIPI DPHY 不同以往平台 MIPI DPHY 版本, 其带宽最高可以到 4.5 Gbps.



应用领域

MIPI DSI基于差分信号传输，可以降低引脚数量和硬件设计复杂度，并保持良好的硬件兼容性。另外，基于MIPI DSI协议的IP还具备低功耗、低EMI的特性。

其应用领域如下图：



- 超高清设备
- 嵌入式显示设备
- 智能座舱
- VR
- 穿戴设备

驱动代码说明：

uboot

驱动位置

```
drivers/video/drm/dw_mipi_dsi2.c
drivers/video/drm/samsung_mipi_dcphy.c
```

驱动配置

```
CONFIG_DRM_ROCKCHIP_DW_MIPI_DSI2=y
CONFIG_DRM_ROCKCHIP_SAMSUNG_MIPI_DCPHY=y
```

kernel

驱动位置

```
MIPI DSI-2 host controller:
drivers/gpu/drm/rockchip/dw-mipi-dsi2-rockchip.c

MIPI DCPHY:
drivers/phy/rockchip/phy-rockchip-samsung-dcphy.c
```

驱动配置

```
cONFIG_ROCKCHIP_DW_MIPI_DSI=y
CONFIG_PHY_ROCKCHIP_SAMSUNG_DCPHY=y
```

参考设备树

DTS 路径：

```
arch/arm64/boot/dts/rockchip/rk3588-evb.dtsi
arch/arm64/boot/dts/rockchip/rk3588-evb1-lp4.dtsi
arch/arm64/boot/dts/rockchip/rk3588-evb2-lp4.dtsi
arch/arm64/boot/dts/rockchip/rk3588-evb3-lp5.dtsi
arch/arm64/boot/dts/rockchip/rk3588-evb4-lp4.dtsi
arch/arm64/boot/dts/rockchip/rk3588s-evb.dtsi
arch/arm64/boot/dts/rockchip/rk3588s-evb1-lp4x.dtsi
arch/arm64/boot/dts/rockchip/rk3588s-evb2-lp5.dtsi
arch/arm64/boot/dts/rockchip/rk3588s-evb4-lp4x.dtsi
```

dts 配置用例场景说明：

```
rk3588-evb1: dsi0->dphy->1080p_panel && dsi1->dphy->1080p_panel;
rk3588-evb2: dsi1->dphy->1080p_panel;
rk3588-evb3: dsi0->dphy->1080p_panel && dsi1->cphy->cphy_panel;
rk3588-evb4: dsi0->dphy->1080p_panel;
rk3588s-evb1: dsi0->dphy->1080p_panel && dsi1->dphy->cmd_no_dsc_panel;
rk3588s-evb2: dsi0->cphy->cphy_panel & dsi1->dphy->1080p_panel;
rk3588s-evb4: dsi0->dphy->1080p_panel && dsi1->dphy->cmd_dsc_panel;
```

屏端配置

DTS 配置

```
dsi0_panel: panel@0 {
    status = "okay";
    compatible = "simple-panel-dsi";
    reg = <0>;
    power-supply = <&vcc3v3_lcd_n>;
    backlight = <&backlight>;
    reset-gpios = <&gpio2 RK_PB4 GPIO_ACTIVE_LOW>;
    reset-delay-ms = <10>;
    enable-delay-ms = <10>;
    prepare-delay-ms = <10>;
    unprepare-delay-ms = <10>;
    disable-delay-ms = <60>;
    dsi,flags = <(MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST |
        MIPI_DSI_MODE_LPM | MIPI_DSI_MODE_EOT_PACKET)>;
    dsi,format = <MIPI_DSI_FMT_RGB888>;
    dsi,lanes  = <4>;
    //phy-c-option;
    //compressed-data;
    //slice-width = <720>;
    //slice-height = <65>;
    //version-major = <1>;
    //version-minor = <1>;

    panel-init-sequence = [
        ...
        05 78 01 11
        05 00 01 29
    ];

    panel-exit-sequence = [
        05 00 01 28
        05 00 01 10
    ];

    disp_timings0: display-timings {
        native-mode = <&dsi0_timing0>;
        dsi0_timing0: timing0 {
            clock-frequency = <132000000>;
            hactive = <1080>;
            vactive = <1920>;
            hfront-porch = <15>;
            hsync-len = <4>;
            hback-porch = <30>;
            vfront-porch = <15>;
            vsync-len = <2>;
            vback-porch = <15>;
            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <0>;
            pixelclk-active = <0>;
        };
    };
```

```
};
```

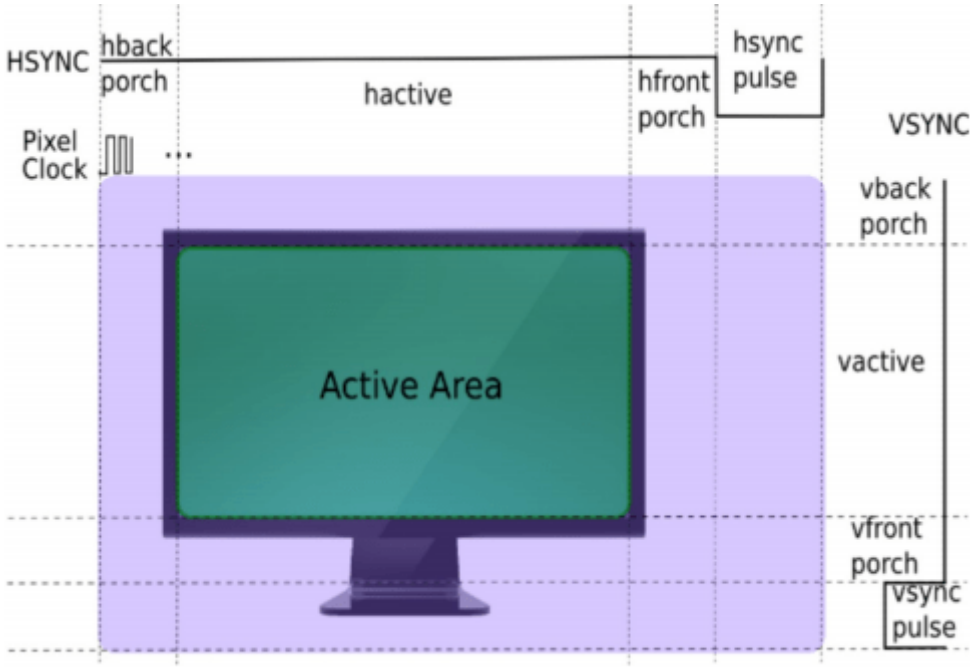
配置说明

通用配置

Property	Description	Value
compatible	compatible	simple-panel-dsi
power-supply	屏端供电 [option]	相关regulator引用
backlight	背光	背光引用
enable-gpios	屏使能GPIO [option]	GPIO引用描述
reset-gpios	屏复位GPIO	GPIO引用描述
reset-delay-ms	panel sequence delay	参考 panel spec
enable-delay-ms		
prepare-delay-ms		
unprepare-delay-ms		
disable-delay-ms		
dsi,flags	DSI2 工作模式	cmd mode: MIPI_DSI_MODE_LPM | MIPI_DSI_MODE_EOT_PACKET
		video mode: MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST | MIPI_DSI_MODE_LPM | MIPI_DSI_MODE_EOT_PACKET
dsi,format	像素数据格式	MIPI_DSI_FMT_RGB888
		MIPI_DSI_FMT_RGB666
		MIPI_DSI_FMT_RGB666_PACKED
		MIPI_DSI_FMT_RGB565
dsi,lanes	mipi data 通道数	1/2/3 trios [cphy]
		6 trios [cphy 双通道]
		1/2/3/4 lanes [dphy]
		8 lanes [dphy 双通道]
phy-c-option	C-PHY panel [option]	布尔类型string
compressed-data	带DSC panel [option]	布尔类型string
slice-width	定义dsc slice宽 [option]	参照panel spec
slice-height	定义dsc slice高 [option]	
version-major	定义dsc版本 [option]	参照panel spec

version-minor		
panel-init-sequence	屏上电初始化序列	[hex] data_type delay_ms payload_lenth payload
panel-exit-sequence	屏下电初始化序列	
display-timing	panel timing	参考panel spec

display Timing

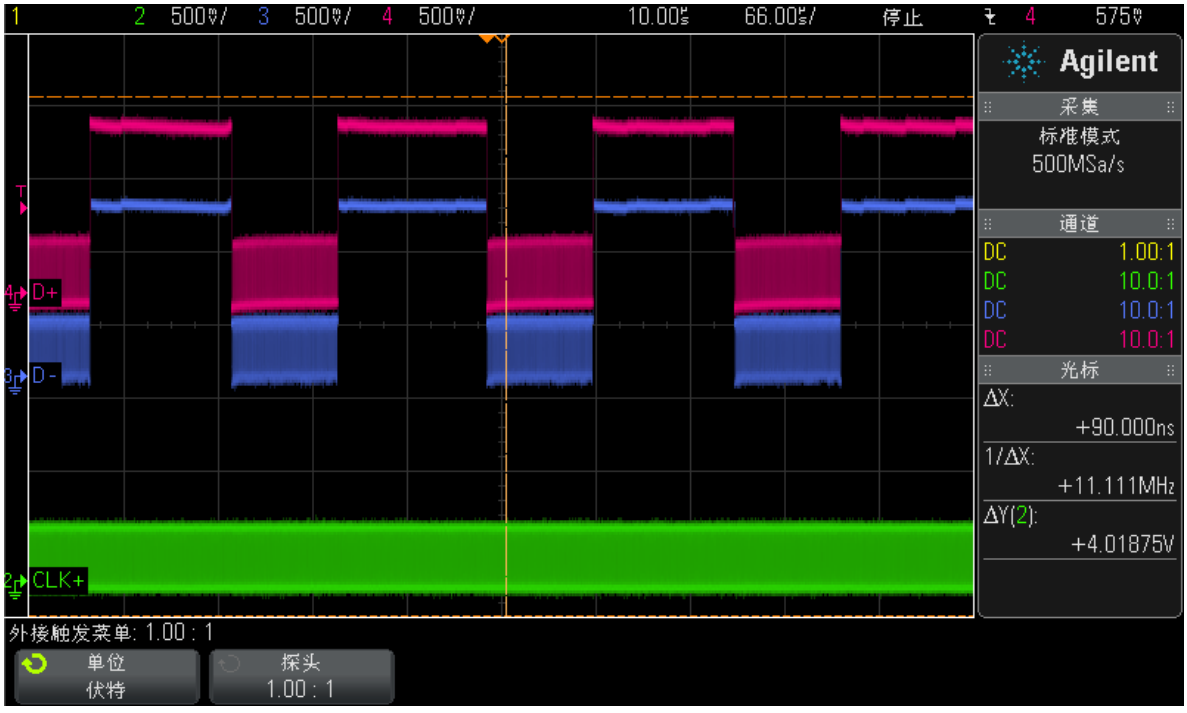


Display Timing

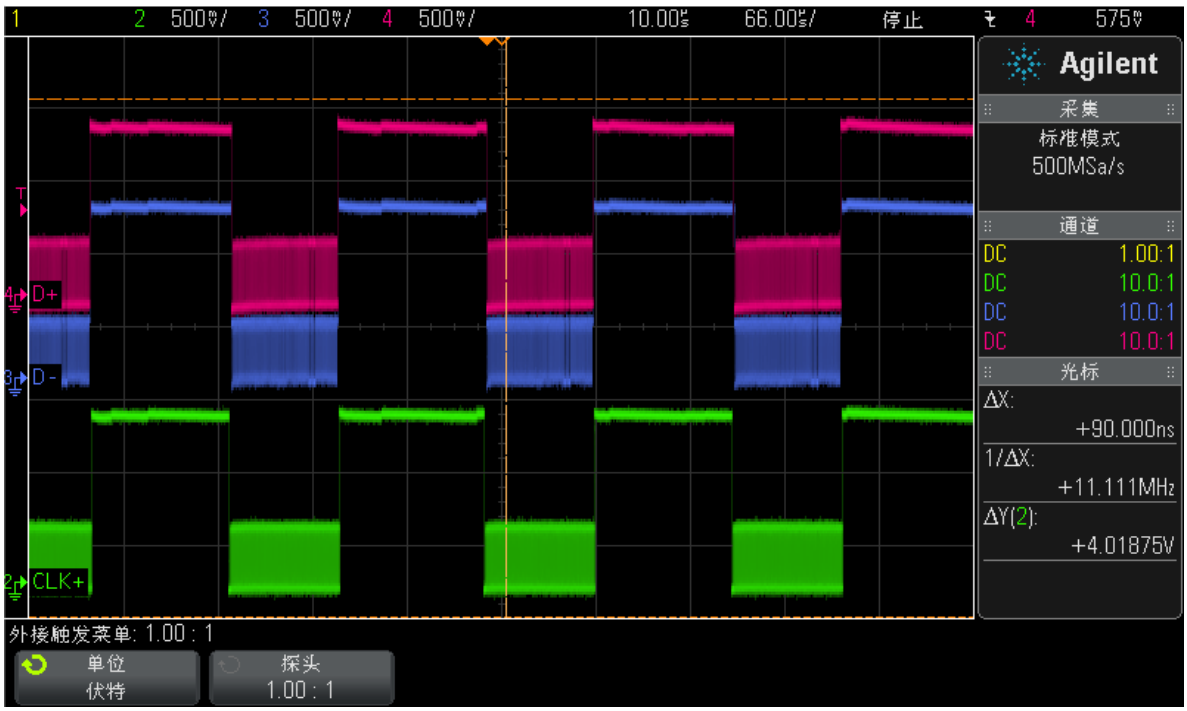
dsi,flags

CLK Type

默认情况， MIPI DPHY 的时钟通道是连续模式，如下图：



当 MIPI_DSI_CLOCK_NON_CONTINUOUS 追加到 dsi,flags 时，MIPI DPHY 的时钟通道将会配置成非连续模式，如下图：



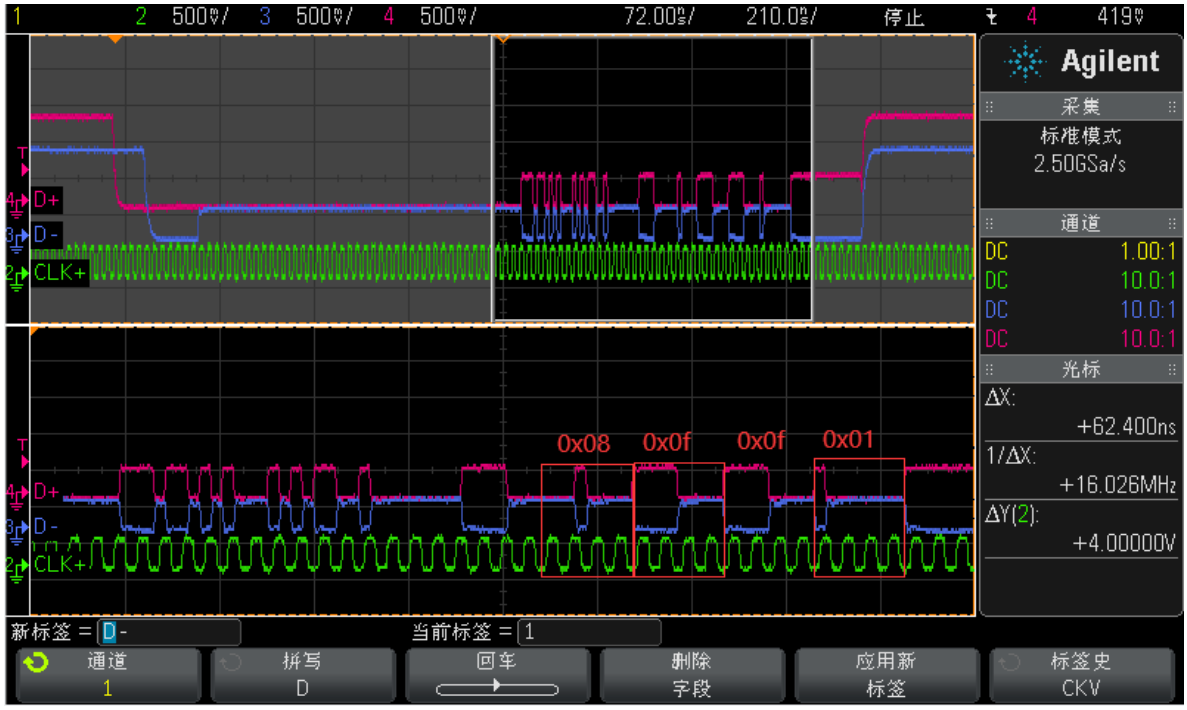
Eotp

Eotp 是一个短包用于指示数据链路上高速传输的结束。Eotp 主要作用是增强系统高速传输通信的稳健性，出于这个目的，DSI 不需要在 LP 模式发送 Eotp。

Eotp 不同于其他 DSI 包，它有固定的格式：

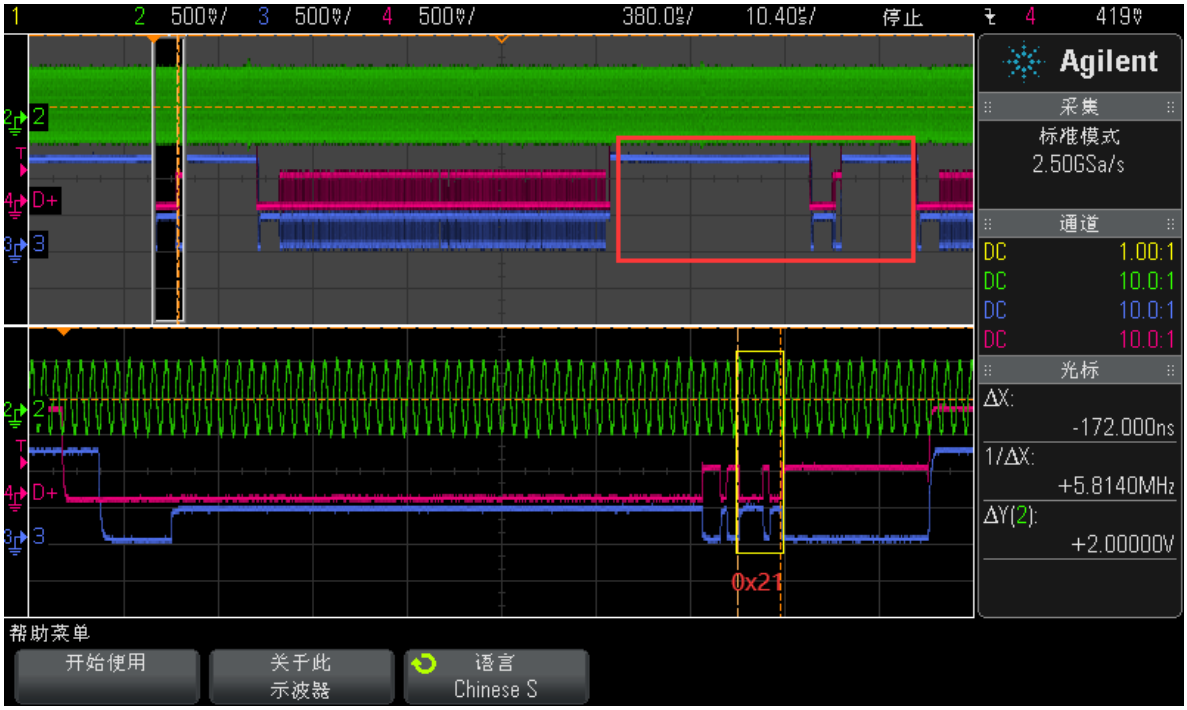
Data Type = DI [5:0] = 0b001000
Virtual Channel = DI [7:6] = 0b00
Payload Data [15:0] = 0x0F0F
ECC [7:0] = 0x01

将 MIPI_DSI_MODE_EOT_PACKET 追加到 dsi,flags 属性可以开关 Soc MIPI DSI TX 在高速模式发送 Eotp。
如下是在 HSDT 模式下捕获 Eotp 波形：



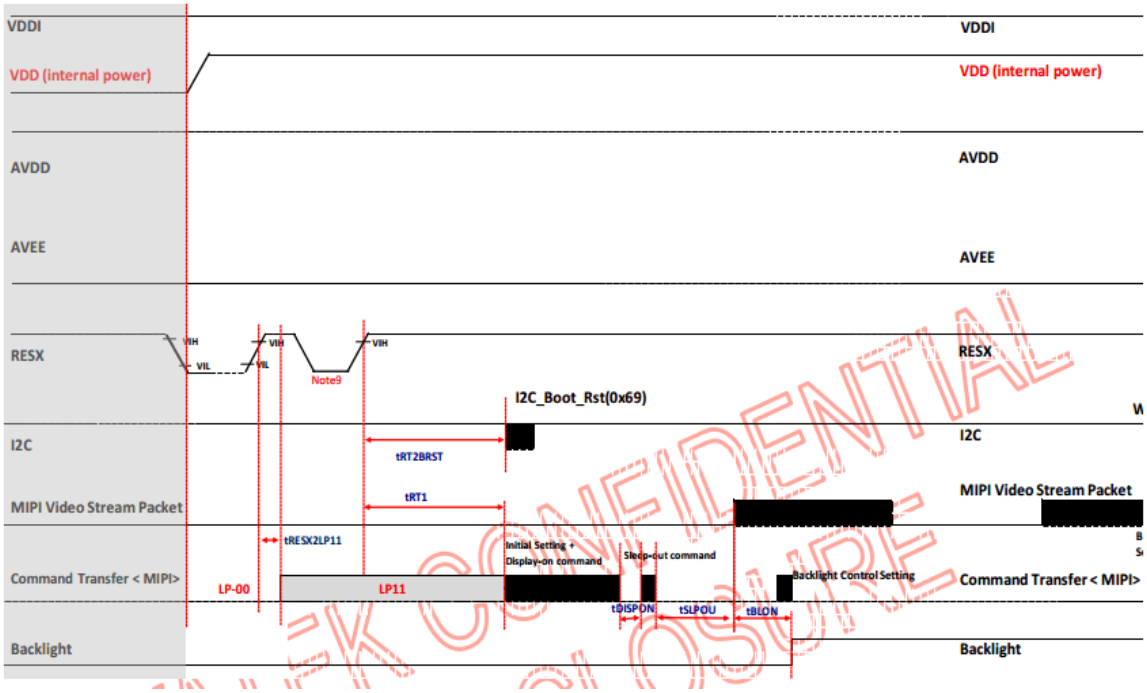
BLANK_HS_EN

在数据通道，一般存在一行会有两个 LP11 消隐, 如下图：

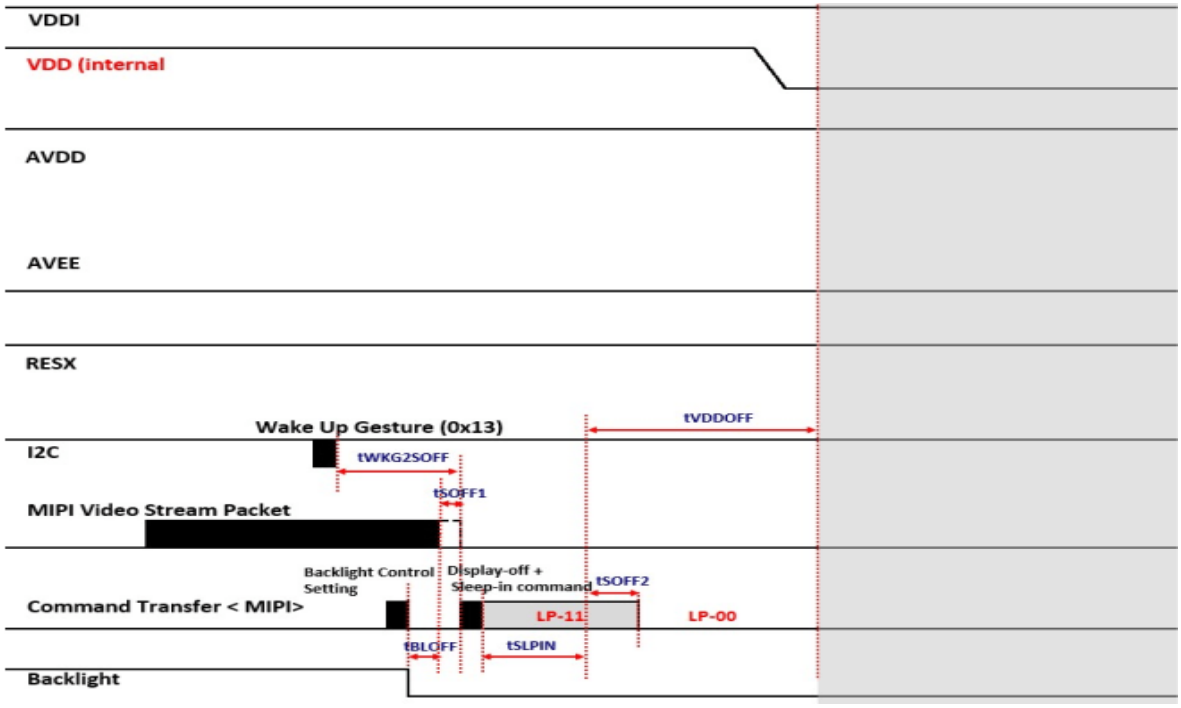


但往往有些显示模组或者外接 MIPI 转接芯片，不支持在 Hblank阶段有两个 LP-11, 可以将 BLK_HFP_HS_EN 或 BLK_HBP_HS_EN 追加到 dsi,flags 属性，使HFP 或 HBP 以高速的形式存在。

屏上电时序



屏下电时序



初始化序列常见数据类型

data type	description	packet size
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
0x05	DCS Short WRITE, no parameters	short
0x15	DCS Short WRITE, 1 parameters	short
0x07	DCS Short WRITE, 1 parameters, DSC EN	short
0x0a	DCS long WRITE, PPS, 128 bytes	long

Bandwidth

MIPI DSI 驱动中会自动按如下公式根据不同的工作模式进行带宽的计算，当然在调试过程中也许对计算的结果想做些微调可以通过 DTS dsi 节点下 rockchip,lane-rate 属性进行指定，单位可以是 Kbps/Mbps(D-PHY) 或 Ksps/Msps (C-PHY)。如下带宽计算用例中可以发现，4K@60 的分辨率对于 RK3588 MIPI DSI DPHY 不需要 DSC 也完全可以支持，但 CPHY 会稍微超出本身最大带宽。

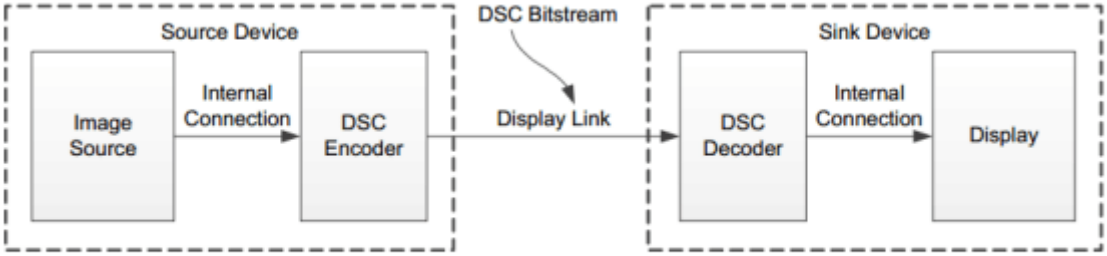
resolution	hact	hfp	hsync	hbp	vact	vfp	vsync	vbp	bpp	lanes
3840x2160	3840	176	88	296	2160	8	10	72	24	4

```
Htotal = hact + hfp + hsync + hbp
Vtotal =  vact + vfp + vsync + vbp
Pixel-CLK = Htotal x Vtotal x 帧率

D-Option:
video burst:
BW = Htotal x Vtotal x 帧率 x BPP x 10 / lanes / 9 (Gbps)

no video burst sync pulse/event:
BW = Htotal x Vtotal x 帧率 x BPP  / lanes (Gbps)

C-Option:
video burst:
BW = Htotal x Vtotal x 帧率 x BPP x 10 / lanes / 9 / 2.28
(Gsps)

no video burst sync pulse/event:
BW = Htotal x Vtotal x 帧率 x BPP  / lanes / 2.28 (Gsps)
```

```
Htotal = 4400
Vtotal = 2250
Pixel-CLK = 4400 x 2250 x 60

D-Option:
video burst:
BW=4400x2250x60x24x10/4/9=3.96 Gbps

no video burst sync pulse/event:
BW=4400x2250x60x24/4=3.57 Gbps

C-Option:
video burst:
BW = 4450x2250x60x24x10/3/9/2.28=2.32 Gbps

no video burst sync pulse/event:
BW = 4450x2250x60x24/3/2.28=2.11 Gbps
```

DSC

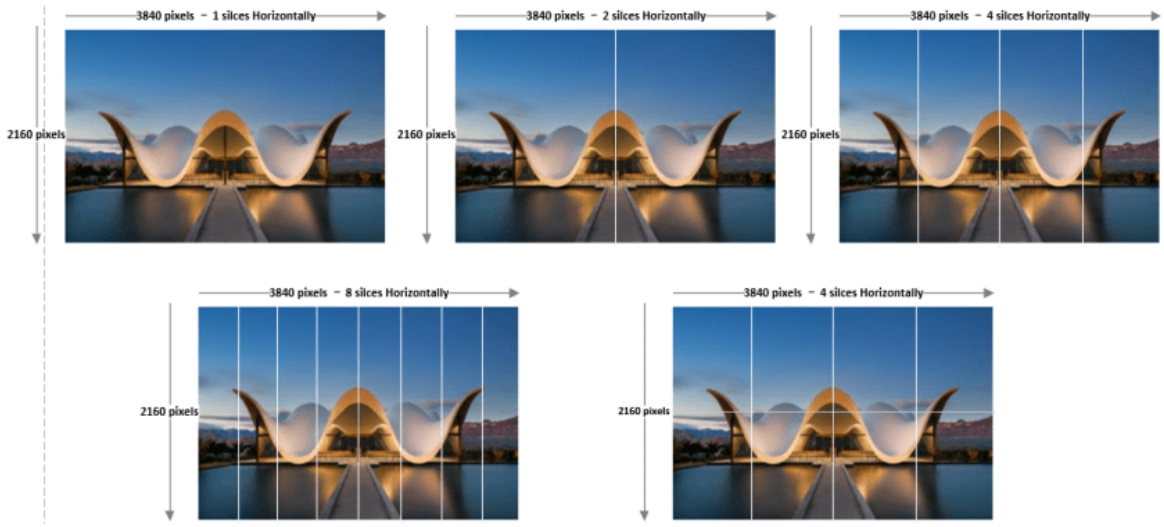
DSC 是 Display Stream Compression 缩写，如下是一个完整的 DSC 系统框图，整个系统为实时工作，未压缩的原始视频像素数据按光栅扫描顺序实时进入编码器并输出比特流，通过显示链路实时传输到解码器，解码器将接收的比特流解码为原始的视频像素数据并送到显示模块显示。从编码器解码输出的图像和输入到编码器前的图像数据信息格式是一样的。RK3588 具有两个 DSC 编码器，可以有效将高分辨率画质内容以低带宽、低延时传输，从而实现MIPI DSI 可以点更高分辨率的画面。



DSC Use in End-to end System

Slice

DSC为加速编码过程，并减少经过压缩失真，DSC导入界面（slice），将每一帧的画面加以切割，切割出的截面同时进行编码。DSC可支持的截面数有1、2、4、8个，甚至更多的截面数。需要注意的是单位为slice/line，line是指画面成形时以raster-scan顺序为一行的像素。除不同截面数外，DSC也可以使用不同长宽的截面。如下图右上及右下两张图片，同样都是一行4个截面，但右上的图切割为长条形截面，右下的图则切割成较窄的长方形截面。要使用哪一种长宽的截面取决于Source Device及Sink Device DSC支持的截面数以及DSC压缩或解压缩率。
RK3588 DSC0 最多支持 8 slices，DSC1 最多支持 2 slices。



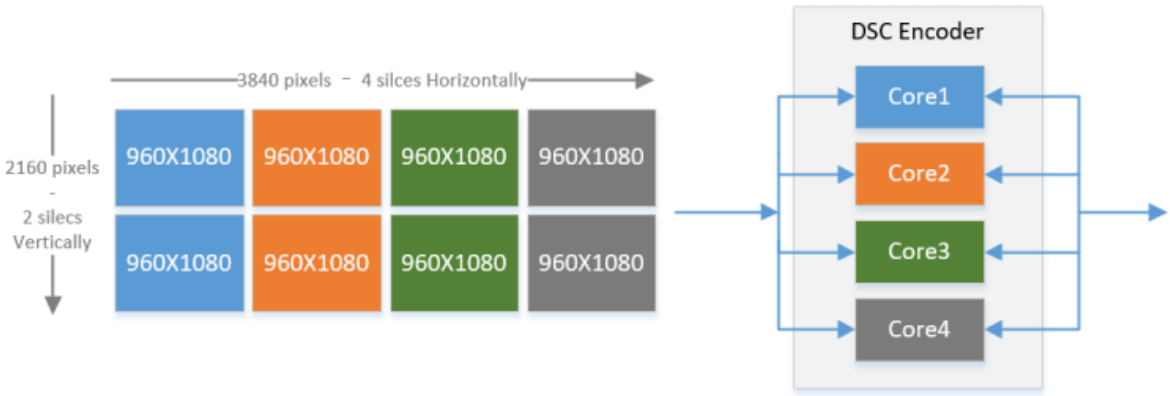
Slice

当实际显示方案中需要确认RK3588 DSC 是否能否支持某款什么分辨率具有多少Slices DSC时，参考如下公式：

```
DSC_8K: active_slice_num * slice_width <= 960 * 8
DSC_4K: active_slice_num * slice_width <= 2048 * 2
```

DSC Encode

在影像数据压缩之前，图像将由Source Device和Sink Device共同协商都能支持的slice数量进行网格均等切割，以3840x2160为例，假如Source Device和Sink Device DSC都能支持8slices，可以在水平方向均等切割4份，垂直方向均等切割2份，假如DSC encoder有4个独立并行处理影像数据压缩的core组成，这该图的左边切割的8个slice可以分成4组同时并行进行压缩，有效将4k画质内容以低带宽、低延时进行传输。



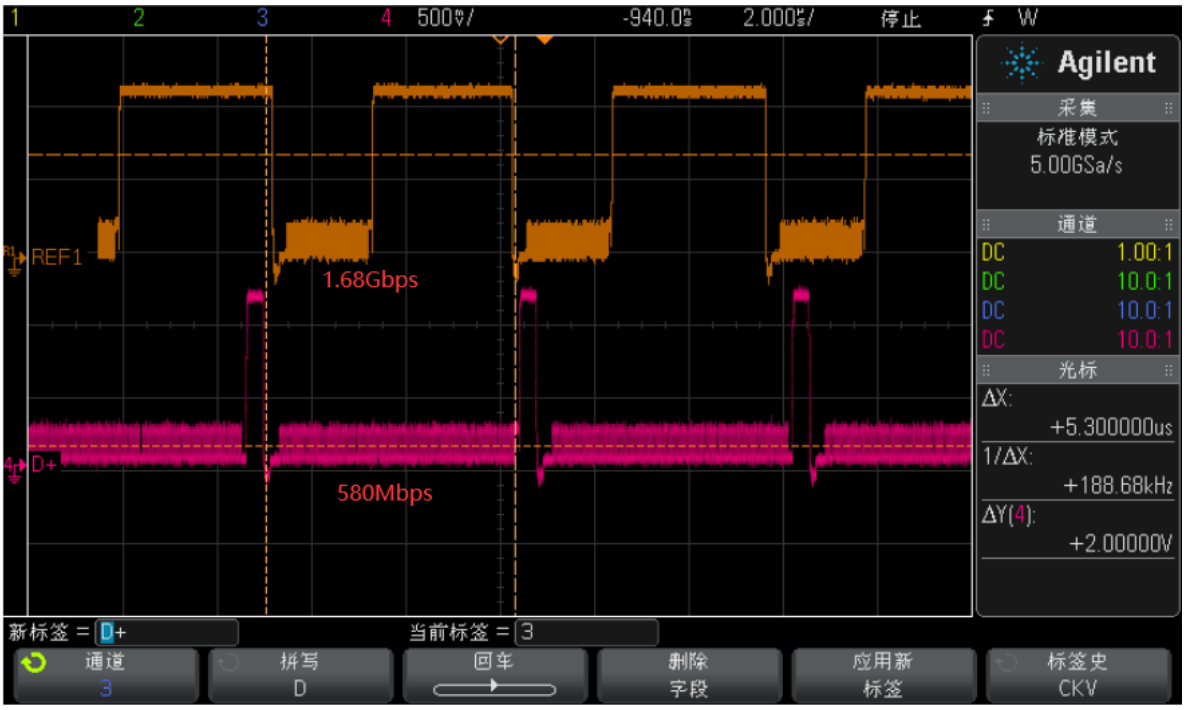
DSC-Encode

DSC Bandwidth

以4Kp60的分辨率为例子，启用DSC前后 MIPI D/C-PHY 带宽变化如下，使能DSC后，链路带宽可以降到原始带宽数据的三分一到二分之一。

resolution	PHY-TYPE	带宽（no dsc）	带宽 (dsc 8bits rgb@60)
3840x2160: 24bits rgb@60	D-PHY （vdieo burst）	3.96 x 4 Gbps	1.32 x 4 Gbsp
	C-PHY （vdieo burst）	2.32 x 3 Gsps	0.774 x 3 Gsps

下图为RK3588 MIPI DSI 驱动 1440x3120p60 4lanes with DSC 显示模组时，分别测量以原始带宽和原始带宽的三分之一的信号波形。



PPS

PPS 一共有128 bytes 长度，如下表会描述一些 DSC 系统需要的重要信息：

1. DSC 版本；
2. DSC 压缩编码前的 BPC；
3. DSC 压缩编码后的 BPP；
4. DSC 压缩编码前输入原始图像的宽高；

5. DSC 压缩编码的 Slice 宽高。

1600 2560 slice 40	Dec	Hex
force reg_dsc_version_major[3:0]	1	1
force reg_dsc_version_minor[3:0]	1	1
force reg_pps_identifier[7:0]	0	00
force reg_bits_per_component[3:0]	8	8
force reg_linebuf_depth[3:0]	9	9
force reg_block_pred_enable	1	1
force reg_convert_rgb	1	1
force reg_simple_422	0	0
force reg_vbr_enable	0	0
force reg_bits_per_pixel[9:0]	128	80
force reg_pic_height[15:0]	2560	0A00
force reg_pic_width[15:0]	1600	0640
force reg_slice_height[15:0]	40	28
force reg_slice_width[15:0]	1600	0640
force reg_chunk_size[15:0]	1600	0640
force reg_initial_xmit_delay[9:0]	512	0200
force reg_initial_dec_delay[15:0]	1057	0421
force reg_initial_scale_value[5:0]	32	0020
force reg_scale_inc_interval[15:0]	1488	05D0
force reg_scale_dec_interval[11:0]	22	0016
force reg_first_line_bpg_offset[4:0]	12	0C
force reg_nfl_bpg_offset[15:0]	631	0277
force reg_slice_bpg_offset[15:0]	218	00DA
force reg_initial_offset[15:0]	6144	1800
force reg_final_offset[15:0]	4320	10E0
force reg_flatness_min_qp[4:0]	3	03
force reg_flatness_max_qp[4:0]	12	0C
	PPS	

	PPS setting	
PA1	11	PPS1
PA2	00	PPS2
PA3	00	PPS3
PA4	89	PPS4
PA5	30	PPS5
PA6	80	PPS6
PA7	0A	PPS7
PA8	00	PPS8
PA9	06	PPS9
PA10	40	PPS10
PA11	00	PPS11
PA12	28	PPS12
PA13	06	PPS13
PA14	40	PPS14
PA15	06	PPS15
PA16	40	PPS16
PA17	02	PPS17
PA18	00	PPS18
PA19	04	PPS19
PA20	21	PPS20
PA21	00	PPS21
PA22	20	PPS22
PA23	05	PPS23
PA24	D0	PPS24
PA25	00	PPS25
PA26	16	PPS26
PA27	00	PPS27
PA28	0C	PPS28
PA29	02	PPS29
PA30	77	PPS30
PA31	00	PPS31

实例

何时启用 DSC

在具体项目 MIPI DSI 相关的显示方案中何时可以启用 DSC。以下面客户提供的一组显示模组参数为例说明启用 DSC 条件：

1. 该模组为一个 DPHY 4 data lanes 接口，最大速率为1Gbps/lane；
2. 该模组为 Video Burst mode, 原始带宽： 1.866Gbps 24 bits RGB 60帧；
3. 该模组支持 DSC 1 Slice
4. 该模组目标速率：676Mbps/lane

4.5.1 Speed Setting

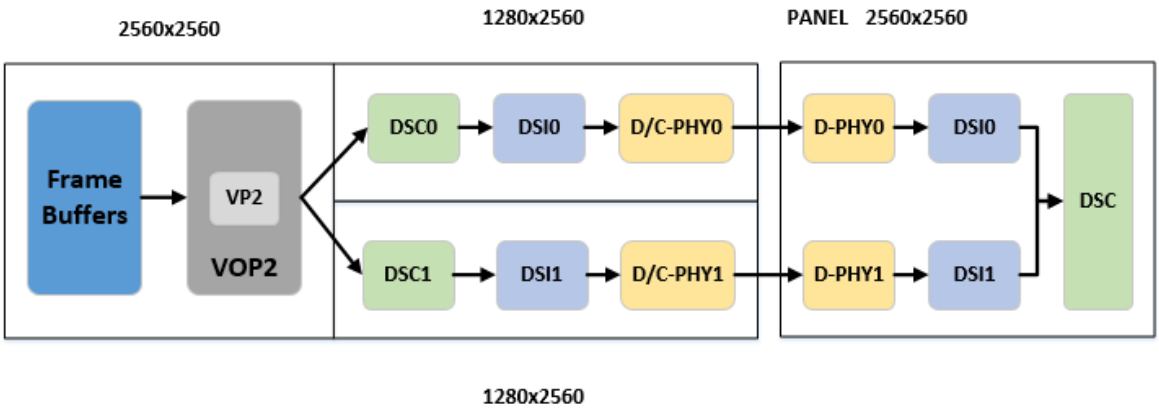
Item	Unit	Value
Frame Rate	Hz	60
Pixel Clock	MHz	84.5
MIPI Lane	Lane	1port* 4Lane
MIPI Speed	sps	676M

4.5.2 Porch Setting

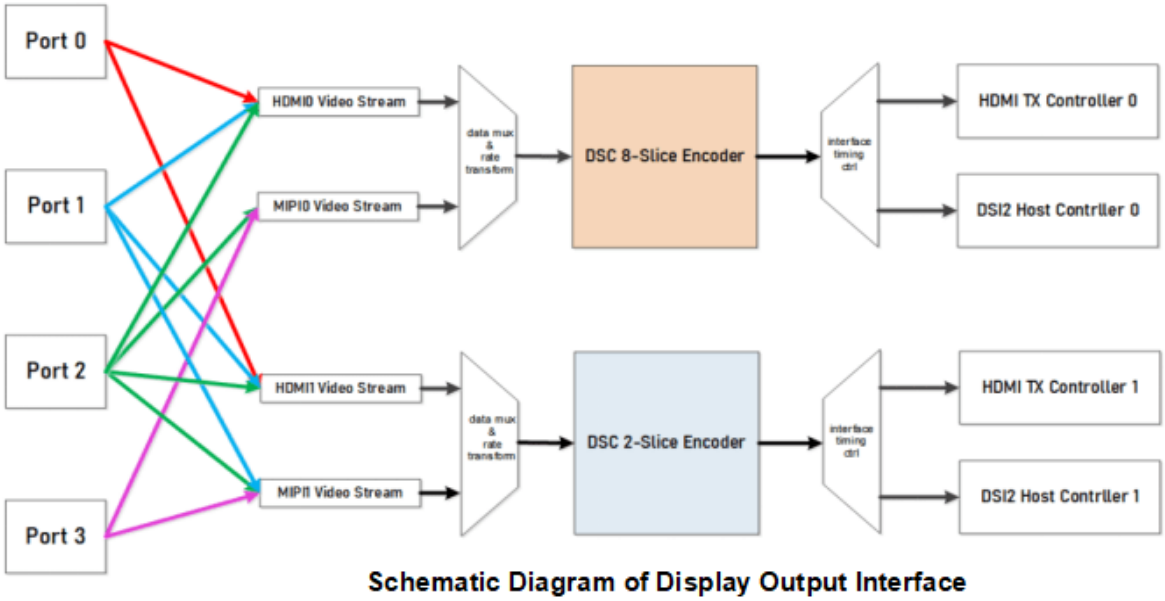
Item	Symbol	Value
Horizontal Sync Width	HSW	20
Horizontal Back Porch	HBP	40
Horizontal Active	HACT	1600
Horizontal Front Porch	HFP	118
Vertical Sync Width	VSW	4
Vertical Back Porch	VBP	18
Vertical Active	VACT	2560
Vertical Front Porch	VFP	60

双通道MIPI 如何启用 DSC

如下图为客户方案中点一款带DSC功能 2560x2560p120 双 mipi 显示模组，其实现的显示链路框图如下。

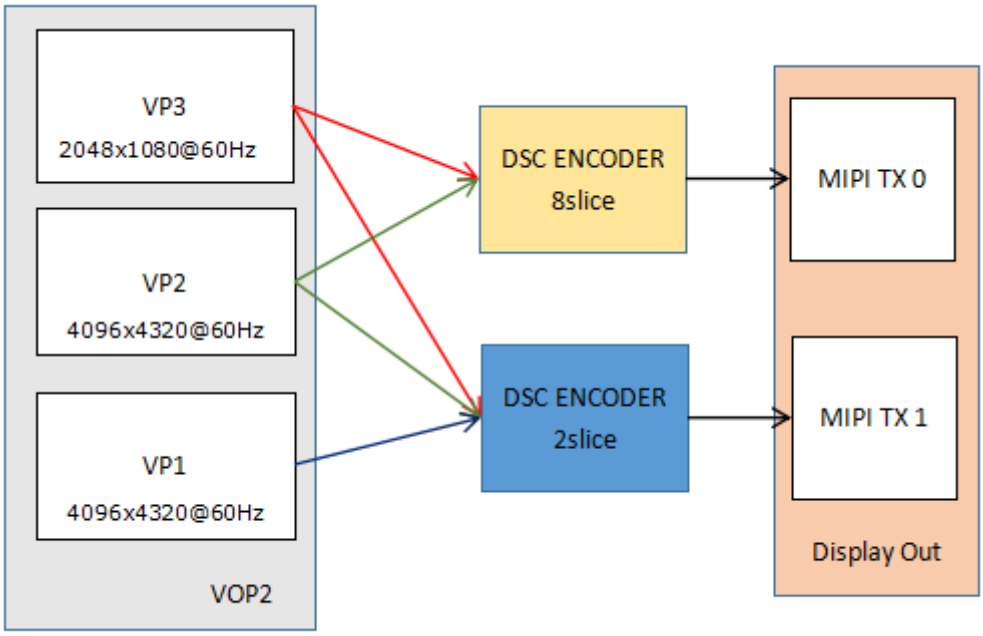


Display Route



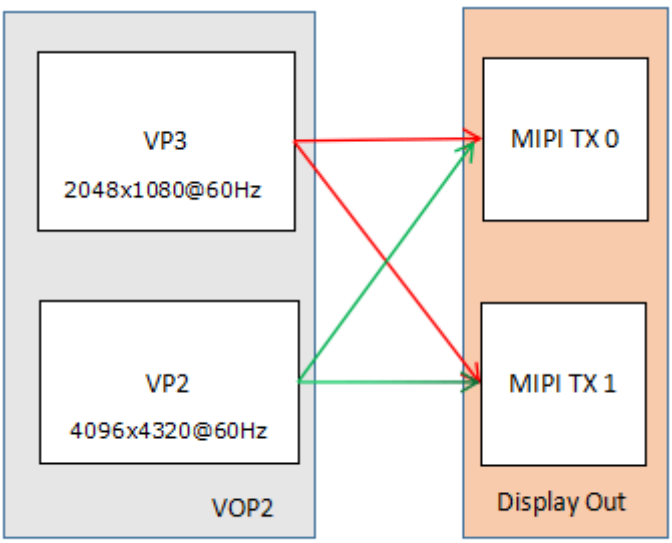
Schematic Diagram of Display Output Interface

MIPI with DSC



Schematic Diagram of Display Output Interface

MIPI with DSC Bypass



Schematic Diagram of Display Output Interface

DTS 配置

如 DSI0 挂载在 VP3:

```
&dsi0_in_vp2 {
    status = "disabled";
};

&dsi0_in_vp3 {
    status = "okay";
};
```

如 DSI1 挂载 VP2:

```
&dsi1_in_vp2 {
    status = "okay";
};

&dsi1_in_vp3 {
    status = "disabled";
};
```

开机LOGO

route_dsi0

例如 vp3->dsi0 或 vp3->dsc0->dsi0:

```
&route_dsi0 {
    status = "okay";
    connect = <&vp3_out_dsi0>;
};
```

route_dsi1

例如 vp2->dsi1 或 vp2->dsc1->dsi1:

```
&route_dsi1 {
    status = "okay";
    connect = <&vp2_out_dsi1>;
};
```

route_dsi0 && route_dsi1

例如 （vp3->dsi0 或 vp3->dsc0->dsi0） && （vp2->dsi1 或 vp2->dsc1->dsi1）：

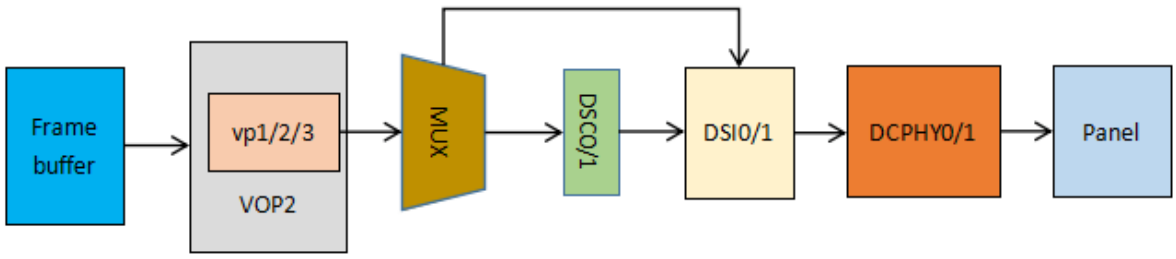
```
&route_dsi0 {
    status = "okay";
    connect = <&vp3_out_dsi0>;
};

&route_dsi1 {
    status = "okay";
    connect = <&vp2_out_dsi1>;
};
```

DSI HOST

ports
以下实例中 ports 是用来 Dispaly Interface 和 panel 之间进行关联。
详细配置说明参阅如下文档:

```
Documentation/devicetree/bindings/graph.txt
```

单 DSI



单 DSI Display Route

DSI0

```
&dsi0 {
    status = "okay";
    //rockchip,lane-rate = <1000>;
    dsi0_panel: panel@0 {
        status = "okay";
        compatible = "simple-panel-dsi";
        ...

        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                panel_in_dsi: endpoint {
                    remote-endpoint = <&dsi_out_panel>;
                };
            };
        };
    };

    ports {
        #address-cells = <1>;
```

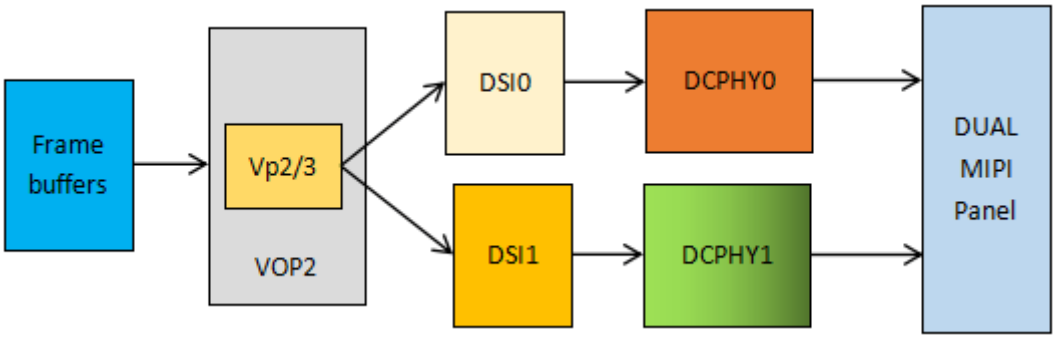
```
        #size-cells = <0>;

        port@1 {
            reg = <1>;
            dsi_out_panel: endpoint {
                remote-endpoint = <&panel_in_dsi>;
            };
        };
    };

};

&mipi_dcphy0 {
    status = "okay";
};
```

DSI1

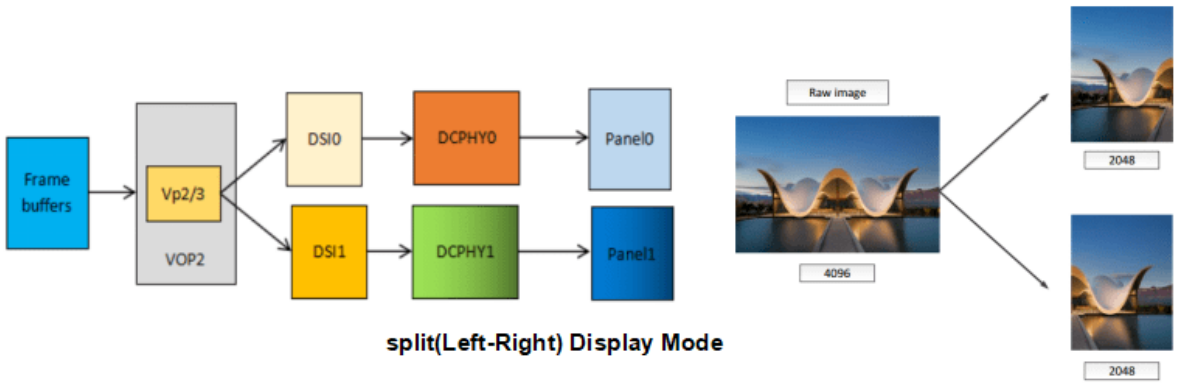
```
&dsi1 {
    status = "okay";
    //rockchip,lane-rate = <1000>;
    dsi1_panel: panel@0 {
        status = "okay";
        compatible = "simple-panel-dsi";
        ...

        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                panel_in_dsi1: endpoint {
                    remote-endpoint = <&dsi1_out_panel>;
                };
            };
        };
    };

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@1 {
            reg = <1>;
            dsi1_out_panel: endpoint {
                remote-endpoint = <&panel_in_dsi1>;
            };
        };
    };
};

&mipi_dcphy1 {
    status = "okay";
};
```

双通道 DSI

MODE1:



MODE2:



双通道的配置注意如下标红属性：

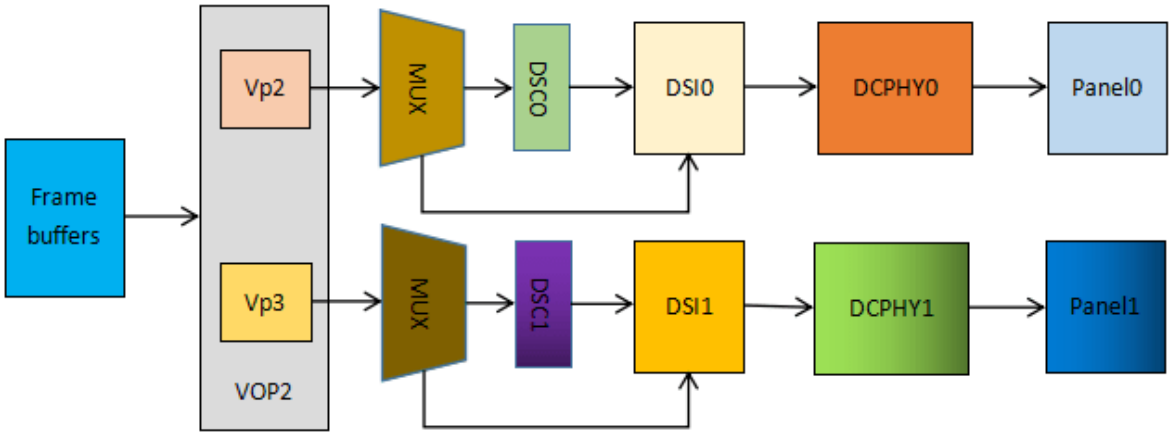
rockchip,dual-channel = <&dsi1>

dsi,lanes = <8>;//DPHY 屏, CPHY 屏值改成 6

```
&dsi0 {
    status = "okay";
    rockchip,dual-channel = <&dsi1>;

    dsi0_panel {
        status = "okay";
        compatible = "simple-panel-dsi";
        dsi,lanes  = <8>;
        ...

        display-timings {
            native-mode = <&timing0>;

            timing0: timing0 {
                clock-frequency = <260000000>;
                hactive = <1440>;
                vactive = <2560>;
                hfront-porch = <150>;
                hsync-len = <30>;
                hback-porch = <60>;
                vfront-porch = <8>;
                vsync-len = <4>;
                vback-porch = <4>;
                hsync-active = <0>;
                vsync-active = <0>;
                de-active = <0>;
```

```
                pixelclk-active = <0>;
            };
        };

        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                panel_in_dsi0: endpoint {
                    remote-endpoint = <&dsi0_out_panel>;
                };
            };
        };
    };

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@1 {
            reg = <1>;
            dsi0_out_panel: endpoint {
                remote-endpoint = <&panel_in_dsi0>;
            };
        };
    };
};

&dsi1 {
    status = "okay";
};

&mipi_dcphy0 {
    status = "okay";
};

&mipi_dcphy1 {
    status = "okay";
};
```

Dual-link DSI



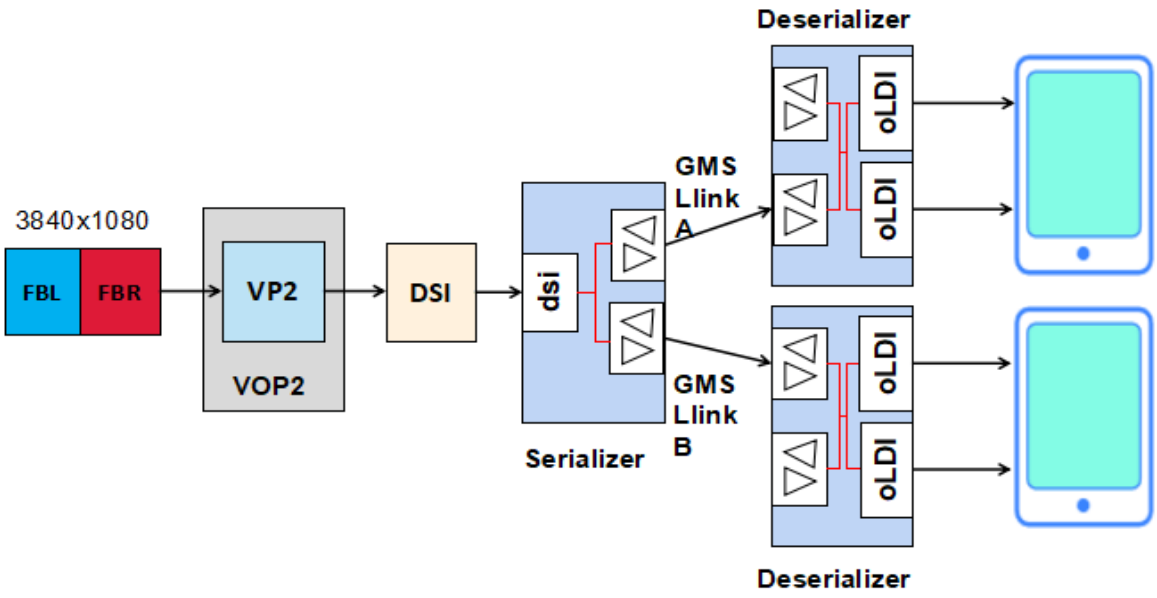
Dual Link DSI Display Route

```
&dsi0 {
    status = "okay";
    //rockchip,lane-rate = <1000>;
    dsi0_panel: panel@0 {
        status = "okay";
        compatible = "simple-panel-dsi";
        ...

        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                panel_in_dsi: endpoint {
                    remote-endpoint = <&dsi_out_panel>;
                };
            };
        };
    };

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@1 {
            reg = <1>;
            dsi_out_panel: endpoint {
                remote-endpoint = <&panel_in_dsi>;
            };
        };
    };

};

&dsi1 { status = "okay";
    //rockchip,lane-rate = <1000>;
    dsi1_panel: panel@0 {
        status = "okay";
        compatible = "simple-panel-dsi";
        ...
```

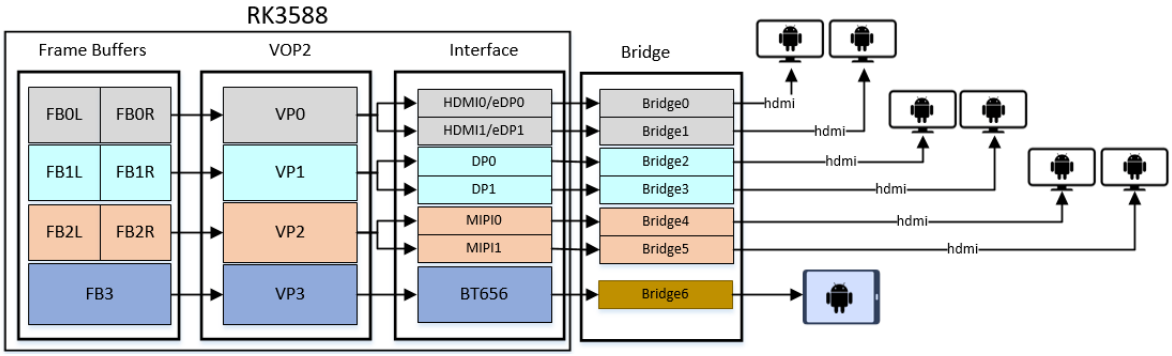
```
        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                panel_in_dsi1: endpoint {
                    remote-endpoint = <&dsi1_out_panel>;
                };
            };
        };
    };

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@1 {
            reg = <1>;
            dsi1_out_panel: endpoint {
                remote-endpoint = <&panel_in_dsi1>;
            };
        };
    };
};

&mipi_dcphy0 {
    status = "okay";
};

&mipi_dcphy1 {
    status = "okay";
};
```

DSI 应用场景

DSI + SerDer 方案



多屏屏接方案



DCPHY

实际应用配置中默认是配置成D-PHY，通过屏端配置介绍可知，通过下面可以配置成 C-PHY:

```
dsi0_panel: panel@0 {
    ...
    phy-c-option;
    ...
};
```

D-PHY

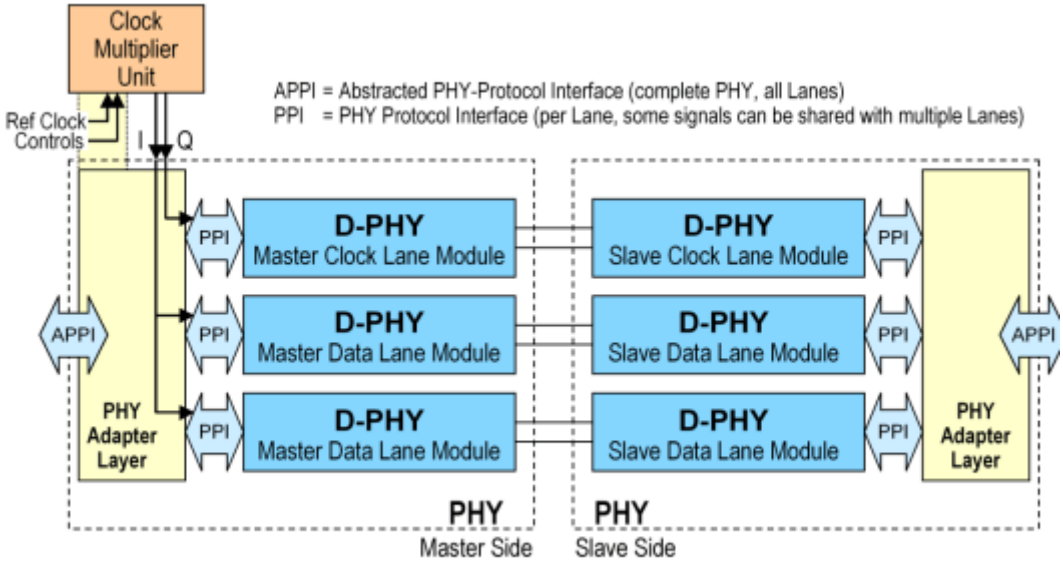


Figure 2 Two Data Lane PHY Configuration

1. Up to 4.5 Gbps per lane in D-PHY
2. 一个D-PHY port 最多4lanes，每个lane由两条差分线组成

C-PHY

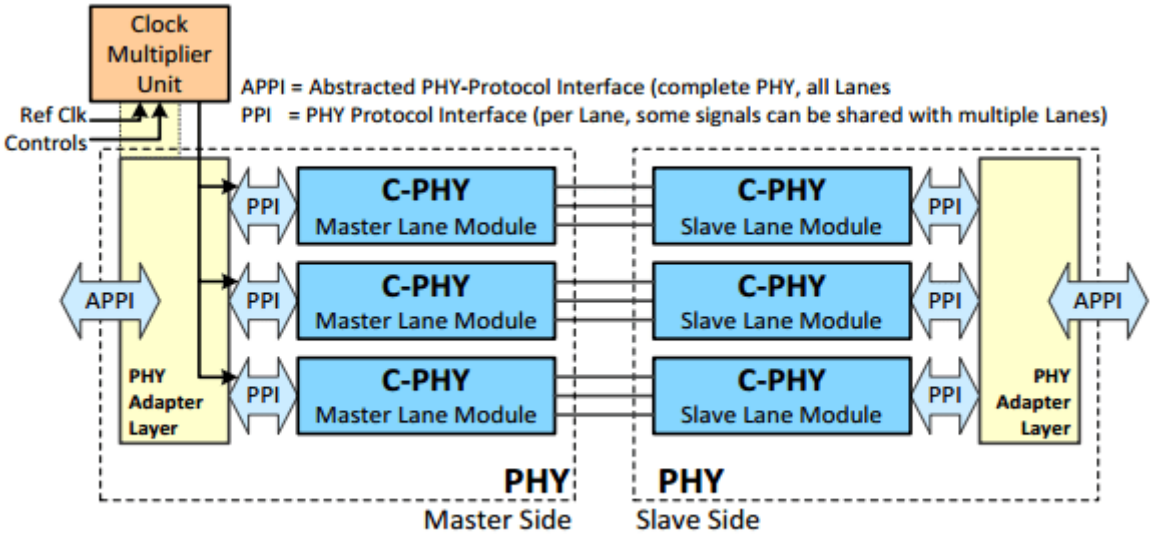
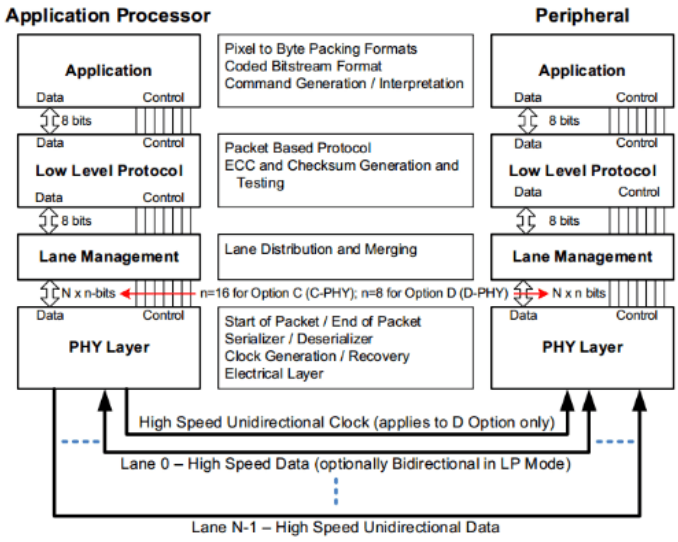


Figure 5 Three Lane PHY Configuration

1. Up to 2.0 Gsps per trio in C-PHY
2. 一个C-PHY port 最多3lanes，每个lane由 tree-wire-trios 组成

协议分析

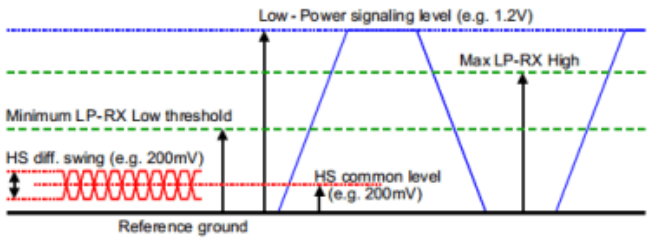
DSI Layer Definitions



- 像素数据打包，编码比特流
- 穿插ECC和CRC
- 像素数据在各个lane上分发和合并
- 物理链路上传输

D Option

Lane states and line levels



Line Levels

- 高速差分信号幅值：200mV
- Low-Power 信号幅值：1.2V

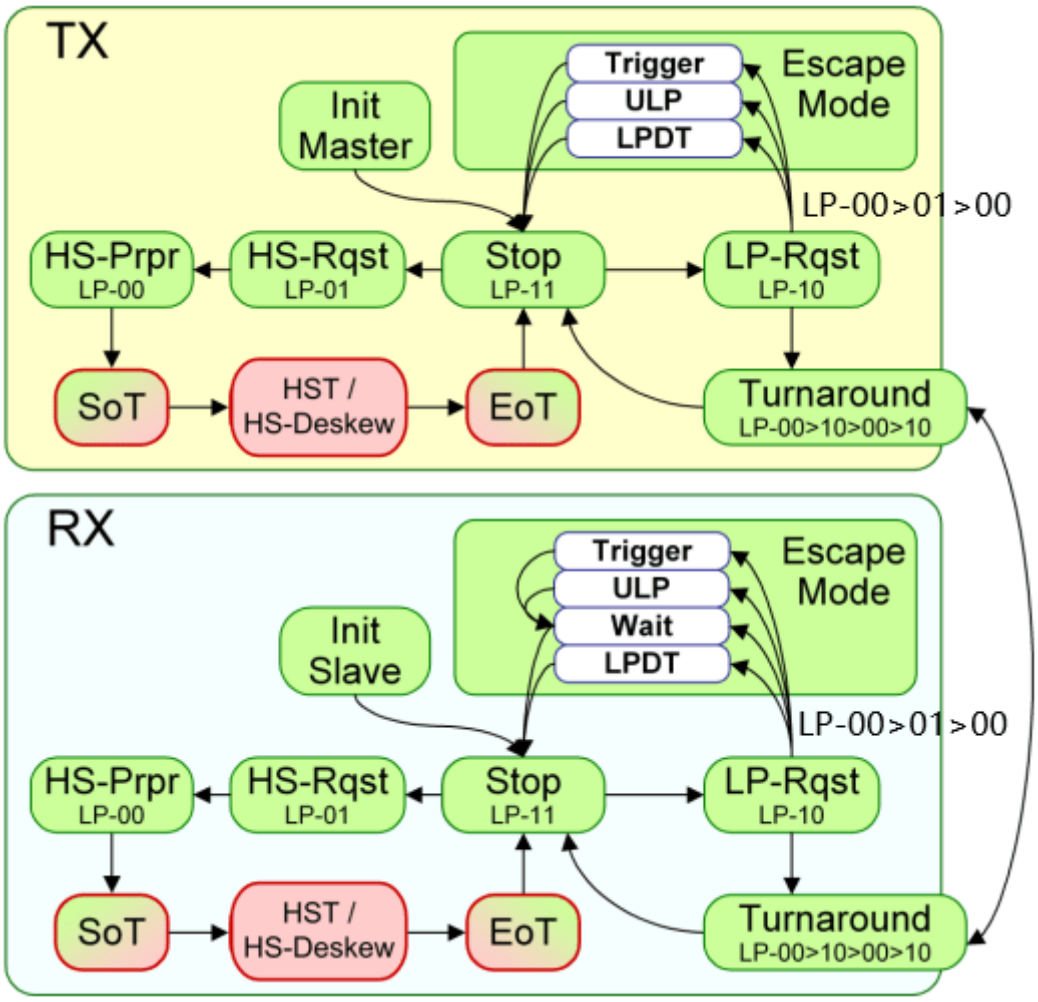
下表罗列在 DPHY Lane 正常操作中可能出现的所有通道状态。

State Code	Line Voltage Levels		High-Speed	Low-Power	
	Dp-Line	Dn-Line	Burst Mode	Control Mode	Escape Mode
HS-0	HS Low	HS High	Differential-0	N/A, Note 1	N/A, Note 1
HS-1	HS High	HS Low	Differential-1	N/A, Note 1	N/A, Note 1
LP-00	LP Low	LP Low	N/A	Bridge	Space
LP-01	LP Low	LP High	N/A	HS-Rqst	Mark-0
LP-10	LP High	LP Low	N/A	LP-Rqst	Mark-1
LP-11	LP High	LP High	N/A	Stop	N/A, Note 2

Note:

1. *During High-Speed transmission the Low-Power Receivers observe LP-00 on the Lines.*
2. *If LP-11 occurs during Escape mode the Lane returns to Stop state (Control Mode LP-11).*

Global Operation Flow Diagram



DSI 数据通道可以驱动到如下三种模式：

1. Escape Mode (只有 Dp0/Dn0 会操作在该模式)；
2. Bus Turnaround Request (只有 Dp0/Dn0 会操作在该模式)；
3. High-Speed Data Transmission。

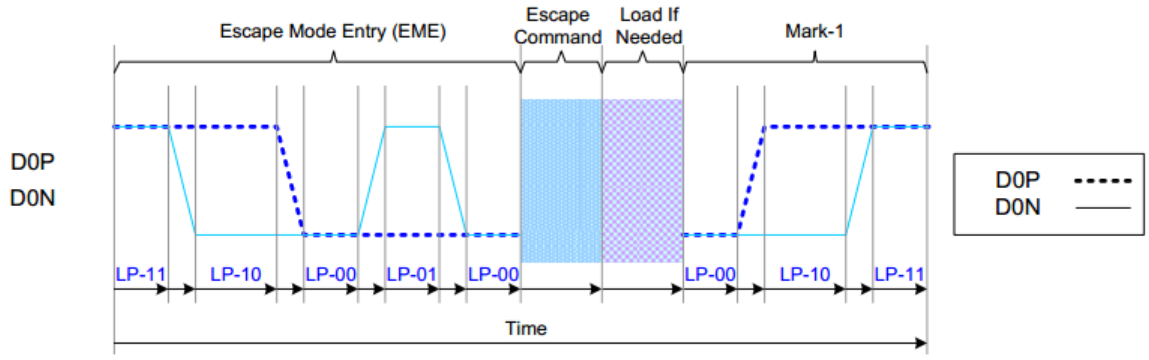
这三种模式和它们进入对应模式的序列定义如下：

Mode	Entering Mode Sequence	Leaving Mode Sequence
Escape Mode[1]	LP-11 → LP-10 → LP-00 → LP-01 → LP-00	LP-00 → LP-10 → LP-11 (Mark-1)
High-Speed Data Transmission[2]	LP-11 → LP-01 → LP-00 → HS-0	(HS-0 or HS-1) → LP-11
Bus Turnaround Request[3]	LP-11 → LP-10 → LP-00 → LP-10 → LP-00	Hi-Z

Entering and Leaving Sequences

Escape Modes

当数据通道处于 LP 模式，数据通道0用于 Escape Mode, 数据通道应通过 LP-11->LP-10->LP-00->LP-01->LP-00 进入 Escape Mode，通过 LP-00->LP-10->LP-11 退出 Escape Mode.



General Escape Mode Sequence

Escape Commands

一旦数据通道进入 Escape 模式， 发送器应该发送 8-bit Escape Comands 指示请求行为，Escape Comands 如下：

Escape Commands

Escape command	Command Type Mode/Trigger	Entry command Pattern (First Bit ➔ Last Bit Transmitted)	Dn	D0
Low-Power Data Transmission	Mode	1110 0001 b	-	X
Ultra-Low Power Mode	Mode	0001 1110 b	X	X
Undefined-1, Note 1	Mode	1001 1111 b	-	-
Undefined-2, Note 1	Mode	1101 1110 b	-	-
Remote Application Reset	Trigger	0110 0010 b	-	X
Acknowledge	Trigger	0010 0001 b	-	X
Unknown-5, Note 1	Trigger	1010 0000 b	-	-

Notes:

1. This Escape command support is not implemented on the display module.

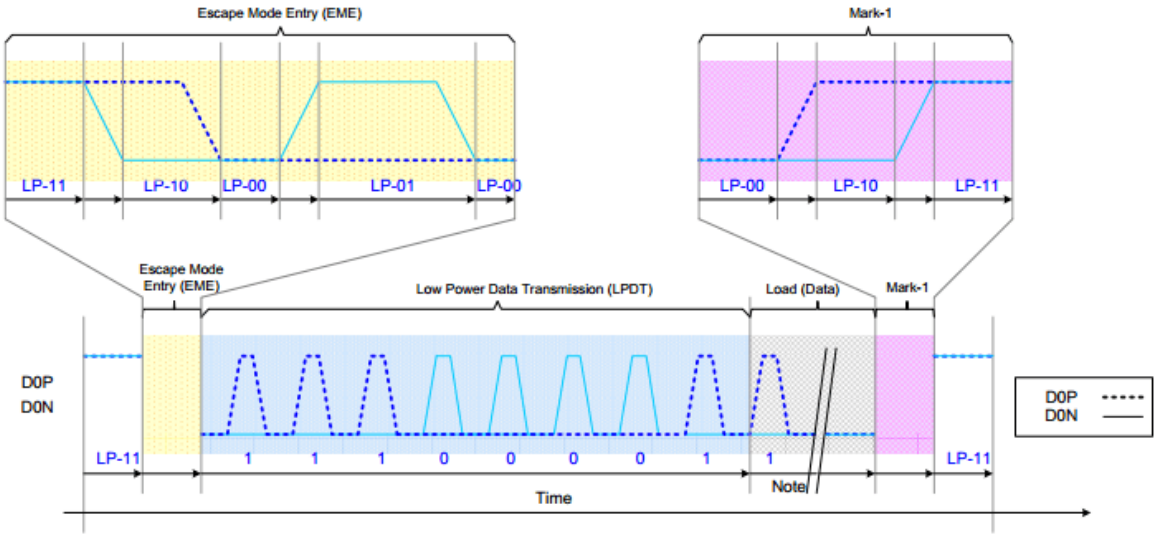
2. n = 1

3. x = Supported

4. - = Not Supported

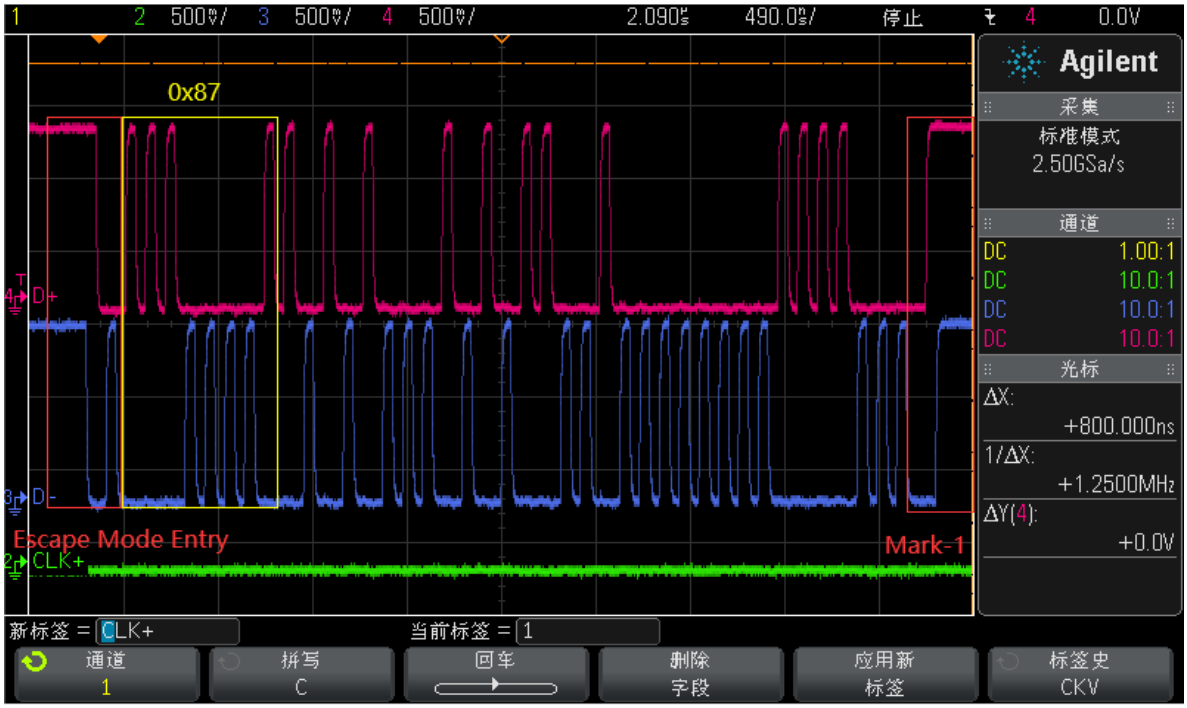
LPDT

当数据通道进入 Escape 模式且向显示模块发送 Low-Power Data Transmission(LPDT) 序列，Soc 的 MIPI DSI TX 可以通过 LPDT 模式向显示模块发送数据, 一般就是通过这种方式向 MIPI 显示模块下载初始化序列。

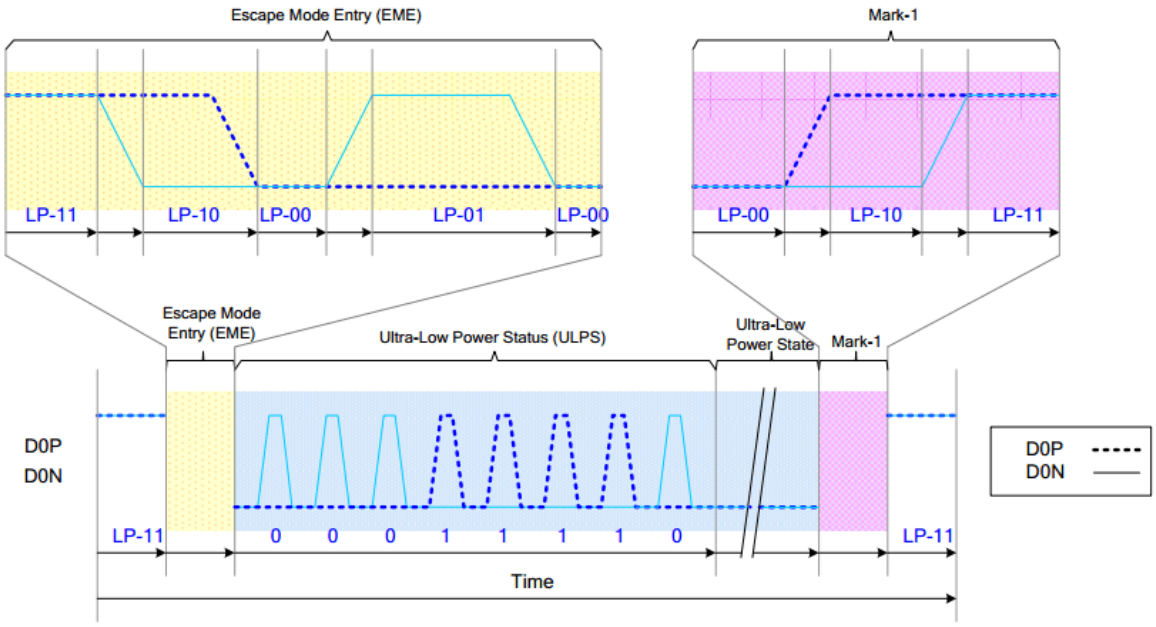


Low-Power DATA Transmission(LPDT)

通过示波器捕获 LPDT 波形如下：



ULPS



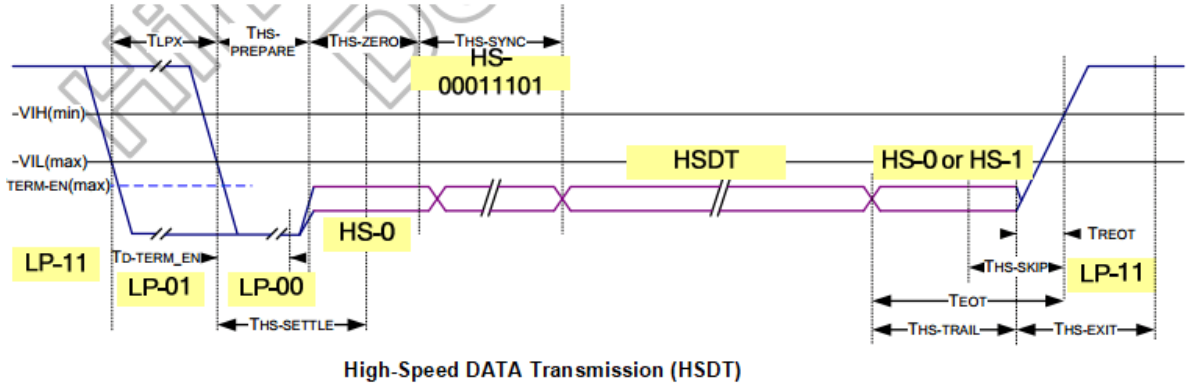
Ultra-Low Power State(ULPS)

HSDT

当 DPHY 的时钟通道在高速时钟模式时，显示模块可以进入高速数据传输模式，所有的数据通道同时进入高速数据传输模式，但可以不同时退出高速模式。数据通道通过如下序列进入高速模式：
LP-11 -> LP-01 -> LP-00 -> HS-0 -> SoT(0001_1101).

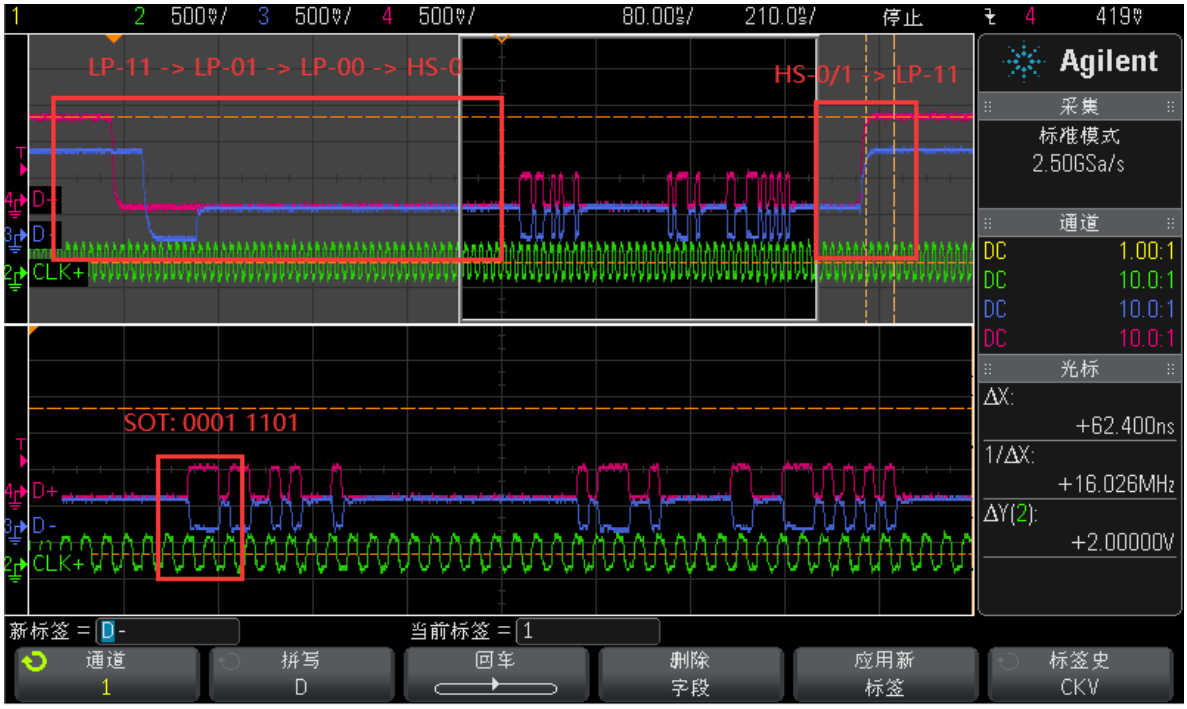
```
1. Start: LP-11
2. HS-Request: LP-01
3. HS-Settle: LP-00 -> HS-0 (RX: Lane Termination Enable)
4. Rx Synchronization: SoT(0001_1101)
5. End: High-Speed Data Transmission (HSDT) - Ready to receive High-Speed Data
Load
```

数据通道退出高速数据传输模式流程：在最后一个有效负载数据之后立即切换差分状态位并保持该状态一段时间 Ths-trail。



High-Speed DATA Transmission (HSDT)

通过示波器捕获 HSDT 波形如下：

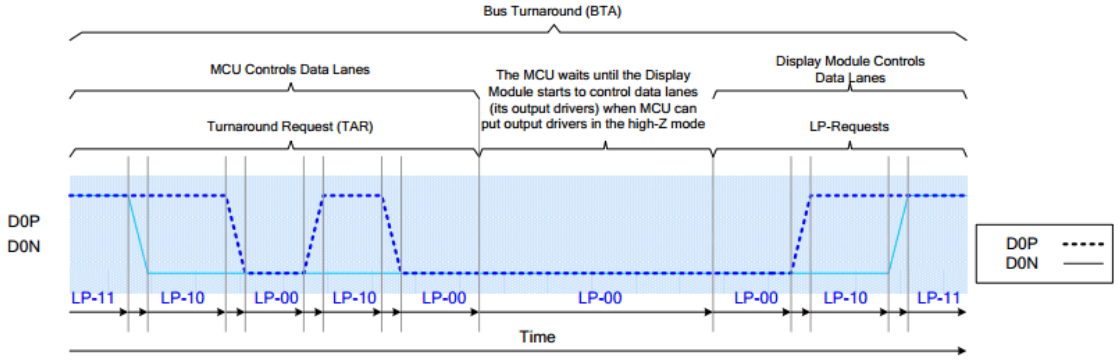


BTA

当需要从显示模块获取信息时，Soc DPHY 的第一数据通道可以通过执行总线翻转步骤。操作步骤如下：

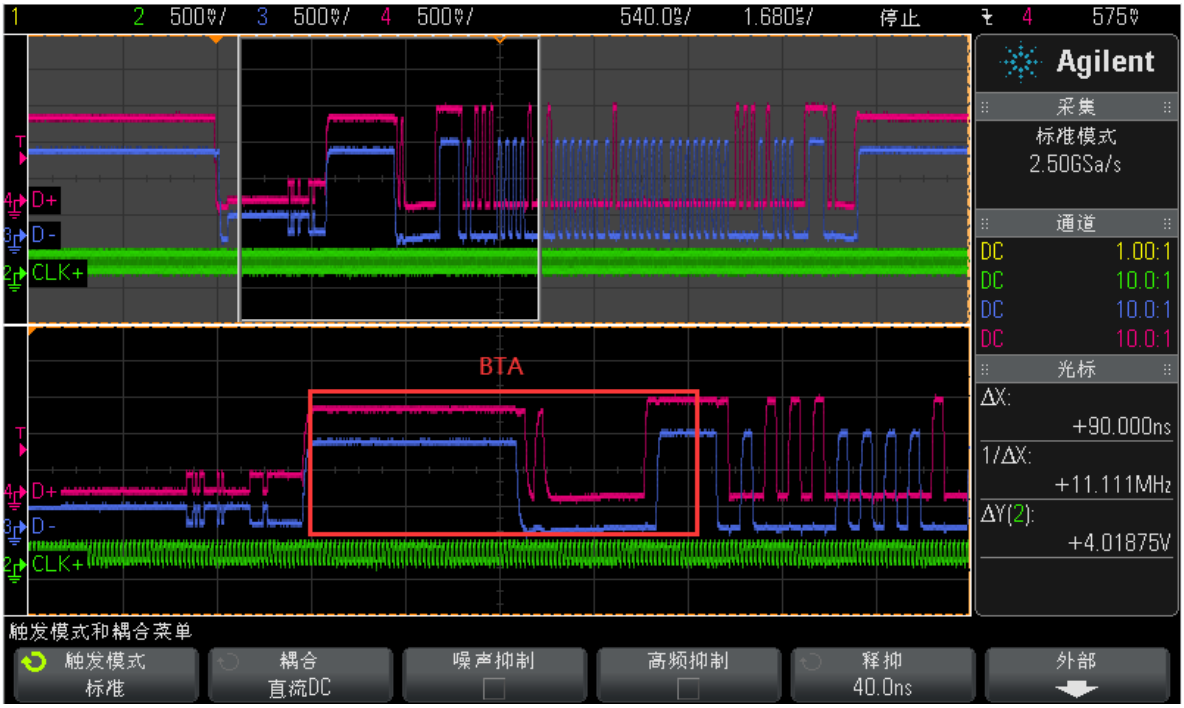
- Start (MCU): LP-11
- Turnaround Request (MCU): LP-11 => LP-10 => LP-00 => LP-10 => LP-00
- The MCU waits until the display module starts to control D0P/N data lanes and the MCU stops to control D0P/N data lanes (= High-Z)
- The display module changes to the stop mode: LP-00 => LP-10 => LP-11

The bus turnaround procedure (from the MCU to the display module) is illustrated below:



Bus Turnaround Procedure

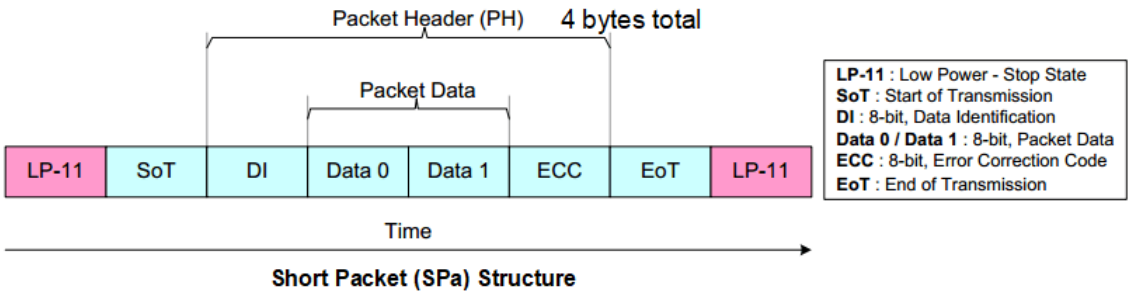
通过示波器在 HSDT 时向显示模块回读并捕获 BTA 波形如下：



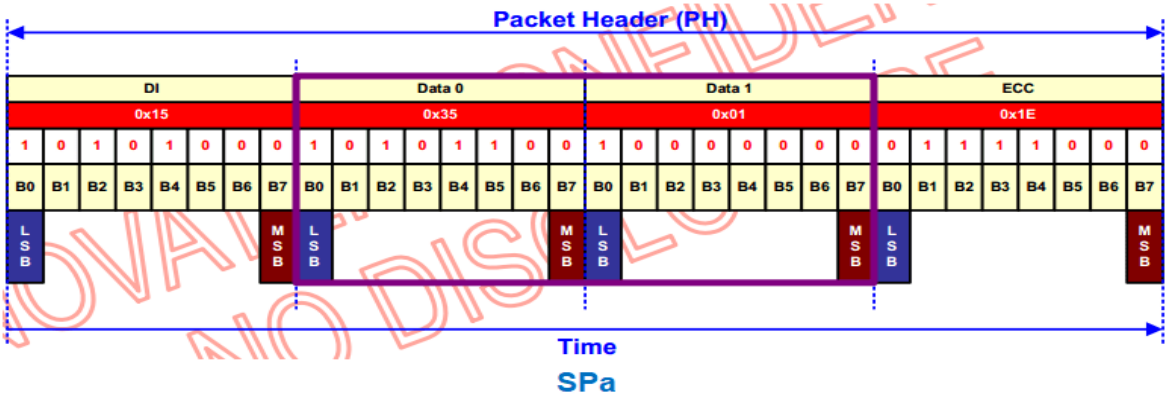
Endian Policy

无论是在 LP 和 HS 数据传输模式，数据都是以长包和短包形式打包并传输给显示模块。

SPa



Example:

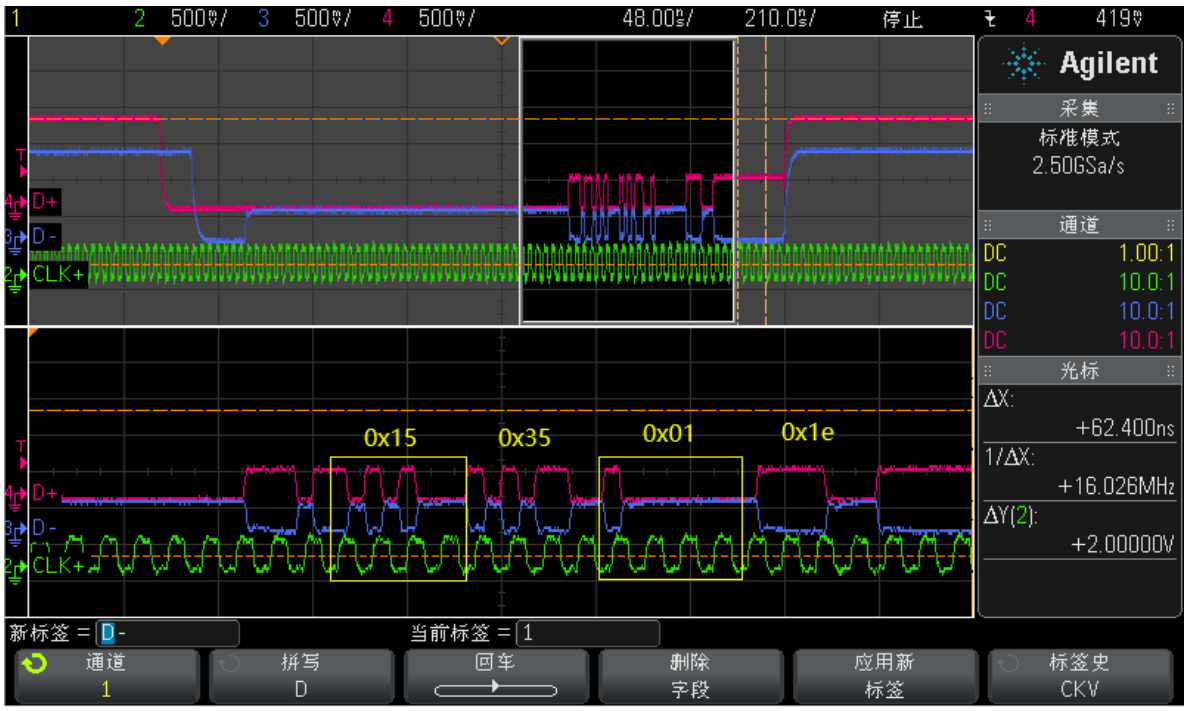


LPDT-SPa

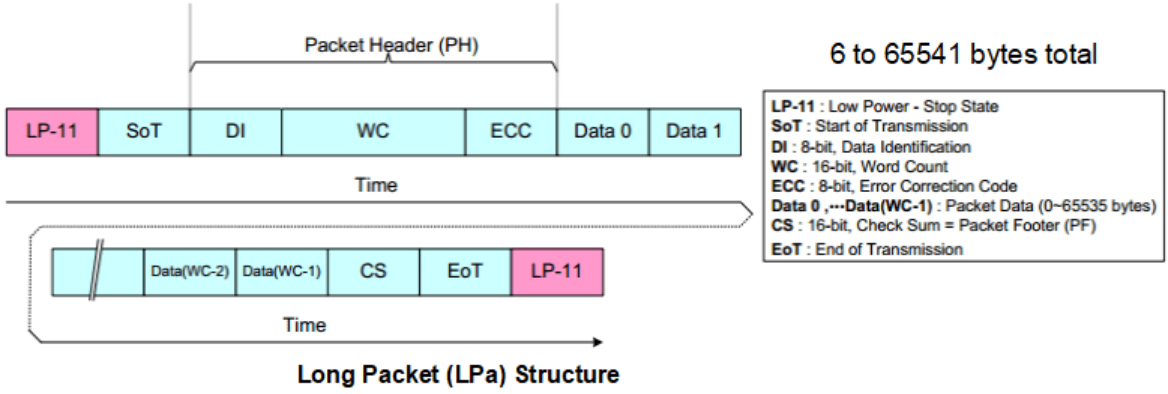
通过示波器在 LPDT 时向显示模块发送如上 SPa 并捕获波形如下：

HSDT-SPa

通过示波器在 HSDT 时向显示模块发送如上 SPa 并捕获波形如下：

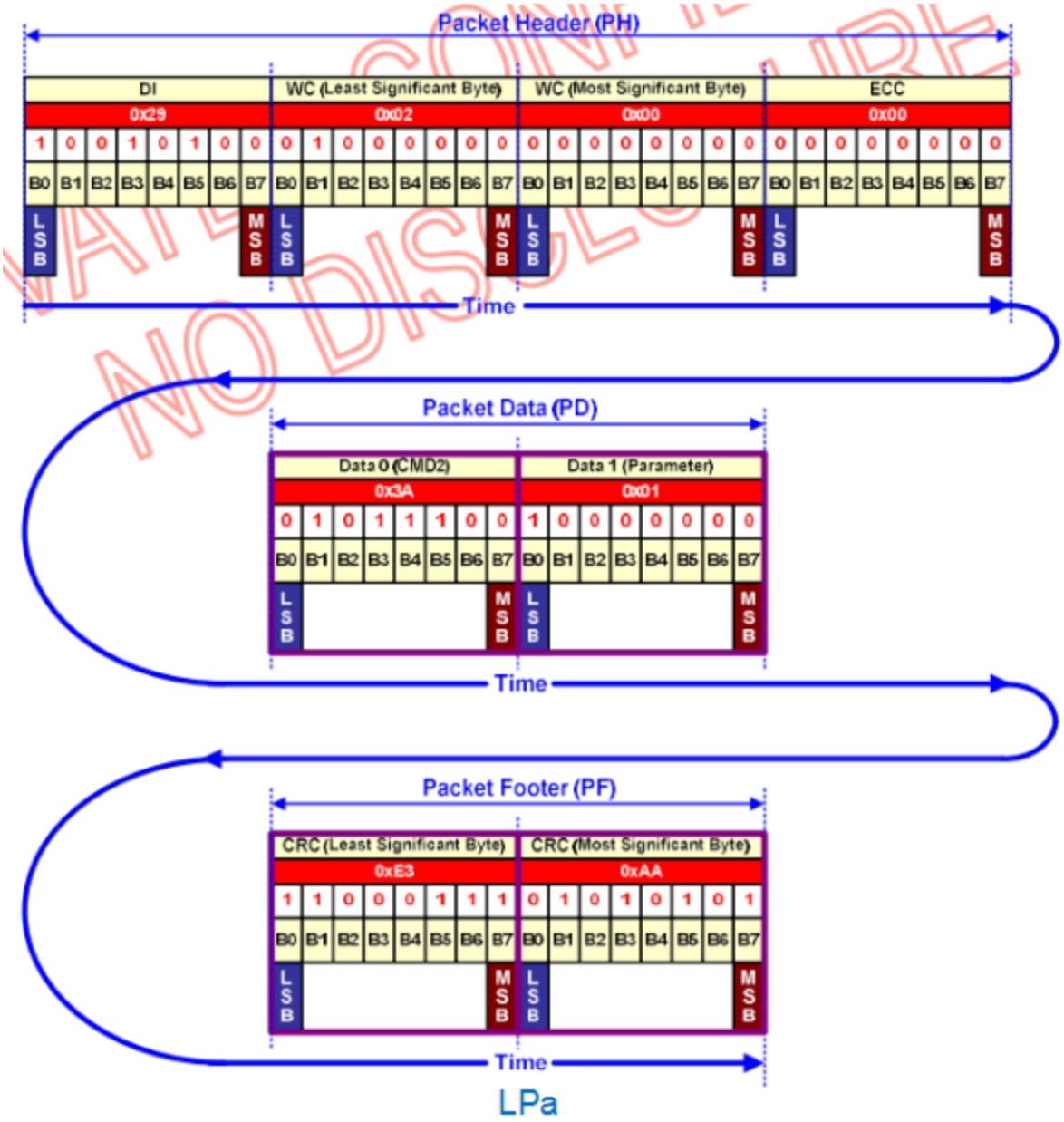


LPa



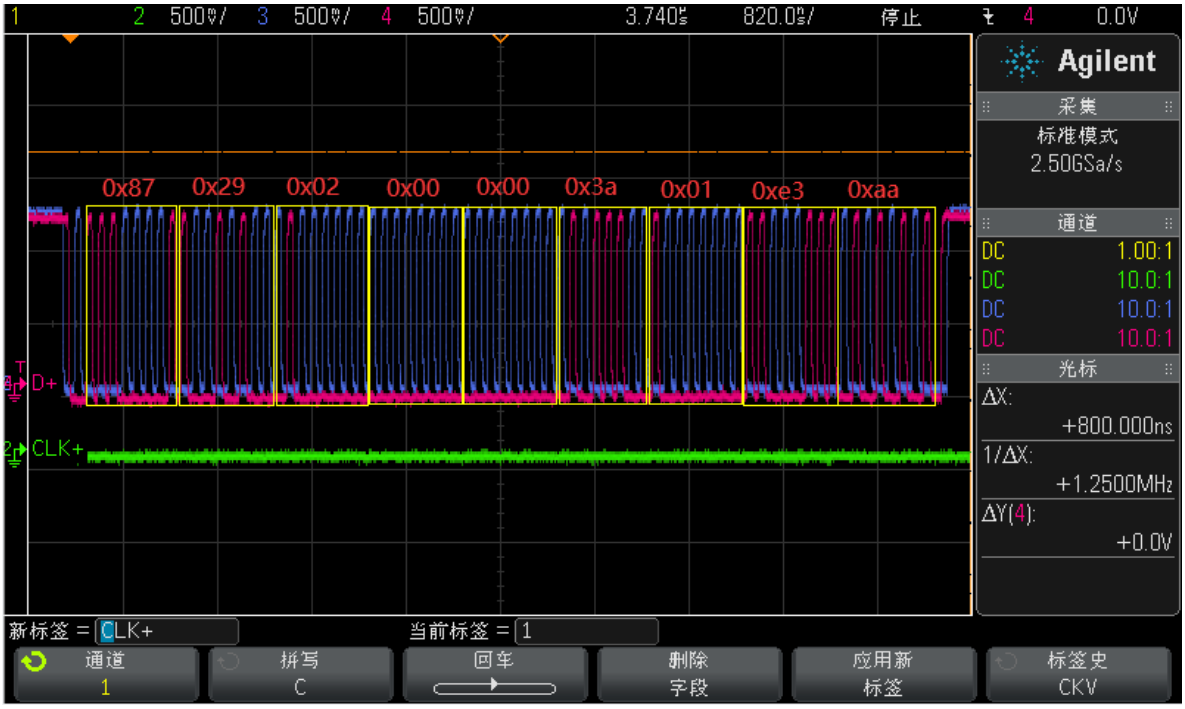
Long Packet (LPa) Structure

Example:



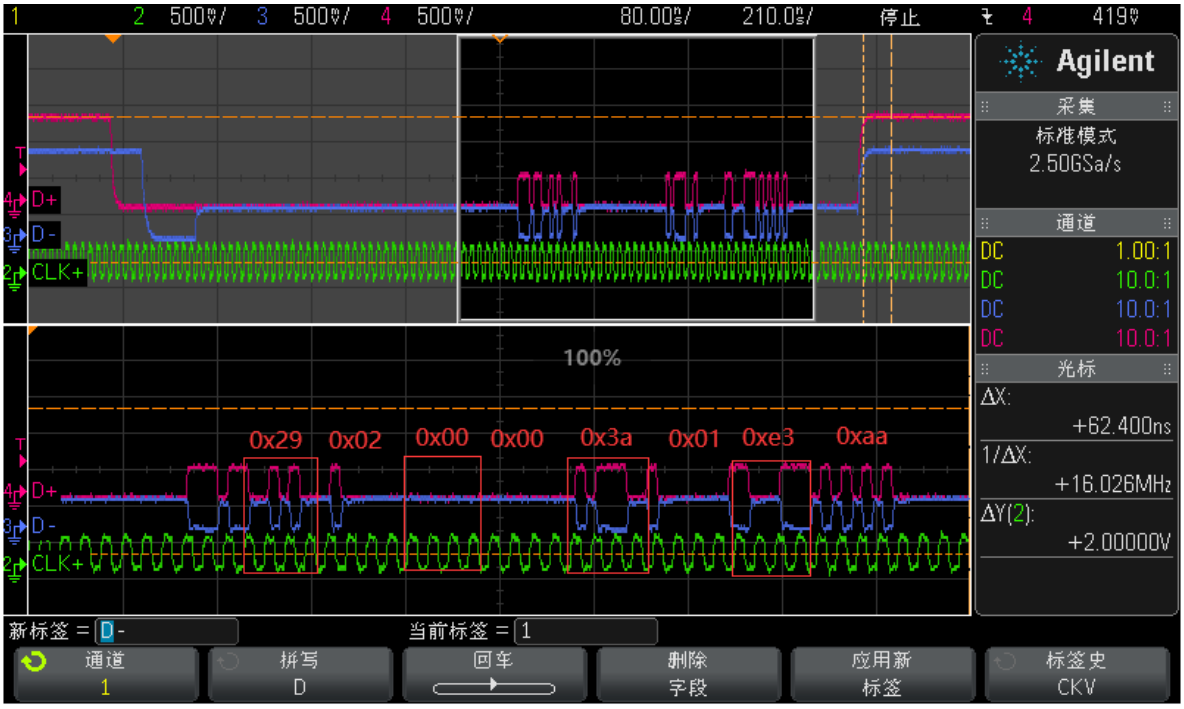
LPDT-LPa

通过示波器在 LPDT 时向显示模块发送如上 LPa 并捕获波形如下：



HSDT-LPa

通过示波器在 HSDT 时向显示模块发送如上 LPa 并捕获波形如下：



DI

如上 SPa 和 LPa 中都有一个 Data Identification (DI), 一个包是长短包就是由 DI 决定，DI 是包头的一部分，由两个部分组成：

- Virtual Channel (VC), 2 bits, DI [7...6]
- Data Type (DT), 6 bits, DI [5…0]

The Data Identification (DI) structure is illustrated, see the figure below.

DI (Data Identification)							
VC (Virtual Channel Identifier)		DT (Data Type)					
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Data Identification (DI) Structure

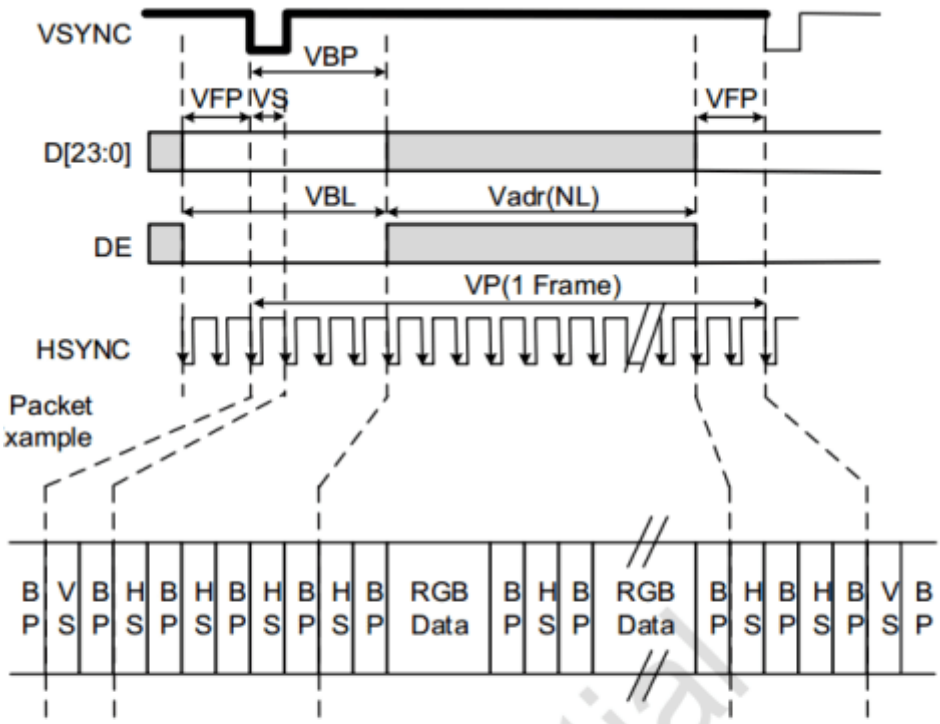
MIPI 协议中目前定义的绝大部分数据类型如下：

Table 16 Data Types for Processor-Sourced Packets

Data Type (hex)	Data Type (binary)	Description	Packet Size
0x01	00 0001	Sync Event, V Sync Start	Short
0x11	01 0001	Sync Event, V Sync End	Short
0x21	10 0001	Sync Event, H Sync Start	Short
0x31	11 0001	Sync Event, H Sync End	Short
0x07	00 0111	Compression Mode Command	Short
0x08	00 1000	End of Transmission packet (EoTp)	Short
0x02	00 0010	Color Mode (CM) Off Command	Short
0x12	01 0010	Color Mode (CM) On Command	Short
0x22	10 0010	Shut Down Peripheral Command	Short
0x32	11 0010	Turn On Peripheral Command	Short
0x03	00 0011	Generic Short WRITE, no parameters	Short
0x13	01 0011	Generic Short WRITE, 1 parameter	Short
0x23	10 0011	Generic Short WRITE, 2 parameters	Short
0x04	00 0100	Generic READ, no parameters	Short
0x14	01 0100	Generic READ, 1 parameter	Short
0x24	10 0100	Generic READ, 2 parameters	Short
0x05	00 0101	DCS Short WRITE, no parameters	Short
0x15	01 0101	DCS Short WRITE, 1 parameter	Short
0x06	00 0110	DCS READ, no parameters	Short
0x16	01 0110	Execute Queue	Short
0x37	11 0111	Set Maximum Return Packet Size	Short
0x27	10 0111	Scrambling Mode Command	Short
0x09	00 1001	Null Packet, no data	Long
0x19	01 1001	Blanking Packet, no data	Long
0x29	10 1001	Generic Long Write	Long
0x39	11 1001	DCS Long Write/write_LUT Command Packet	Long
0x0A	00 1010	Picture Parameter Set	Long
0x0B	00 1011	Compressed Pixel Stream	Long
0x0C	00 1100	Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format	Long

Video Mode Interface Timing

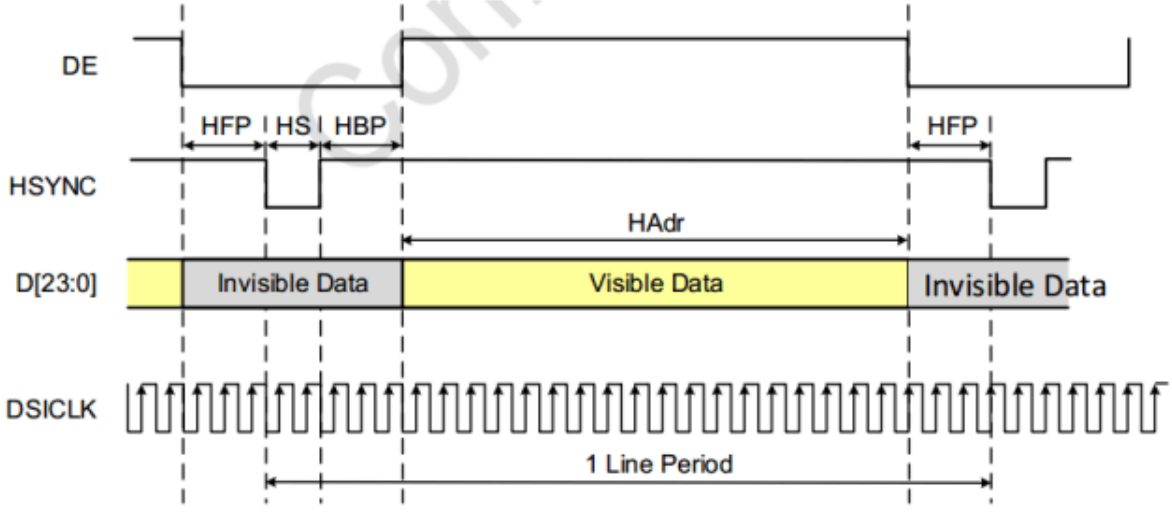
Vertical Display Timing (Video mode)



Key:
VS: Vsync Start (DT = 01h), **HS**: Hsync Start (DT = 21h),
BP: Blanking Packet (DT = 09h), **RGB Data**: RGB Data (DT = 3Eh),
DT: Data Type

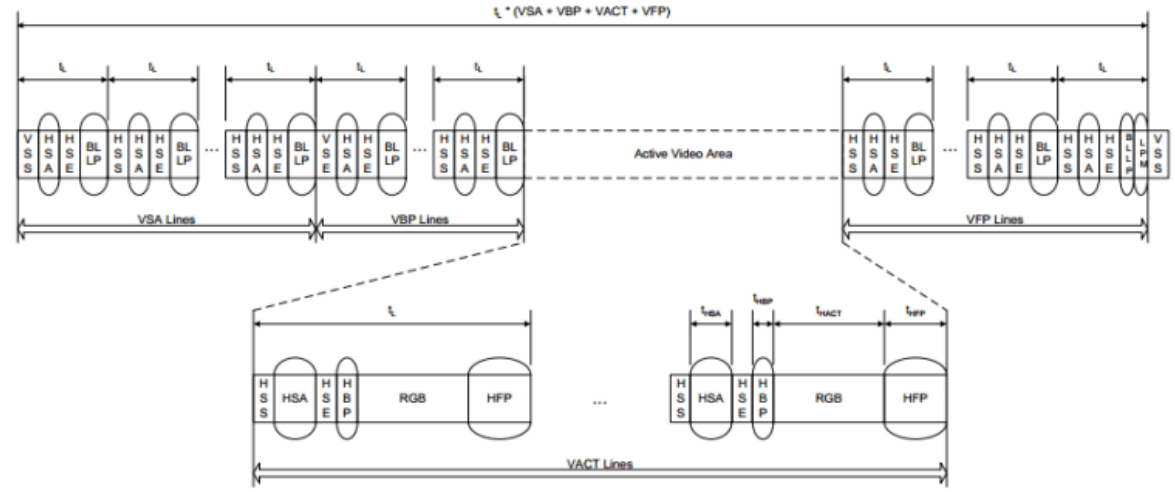
Vertical display timing (video mode)

Horizontal Display Timing (Video mode)



Horizontal display timing (video mode)

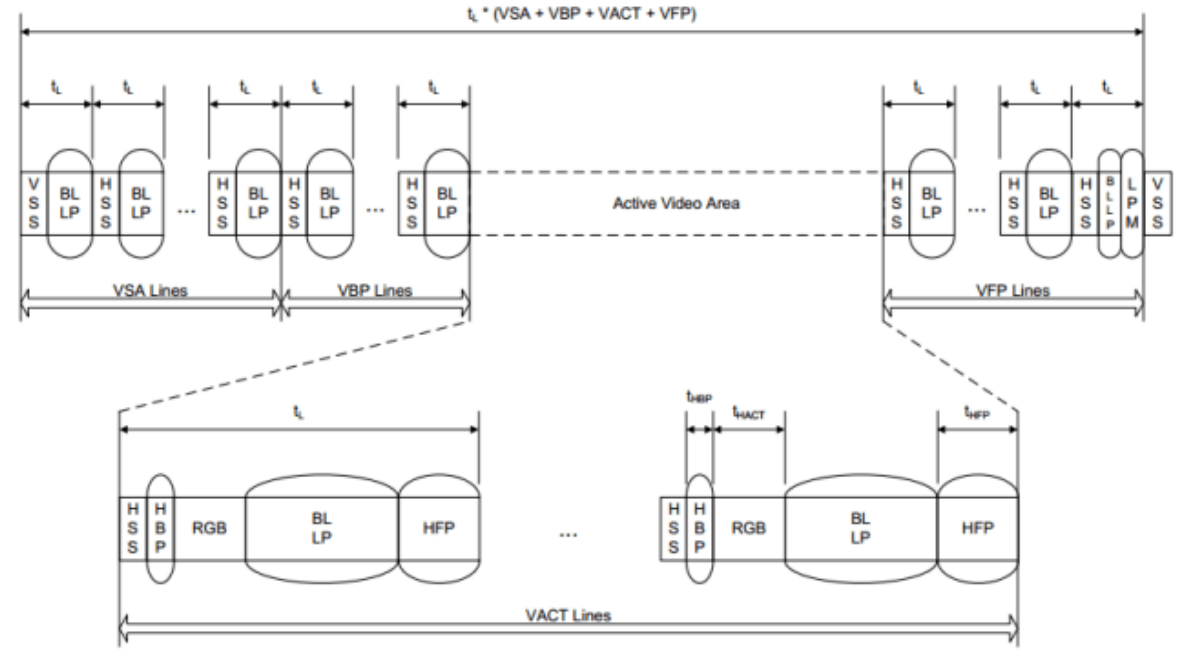
Non-Burst Mode with Sync Pulses



Non-Burst Mode with Sync Pulses

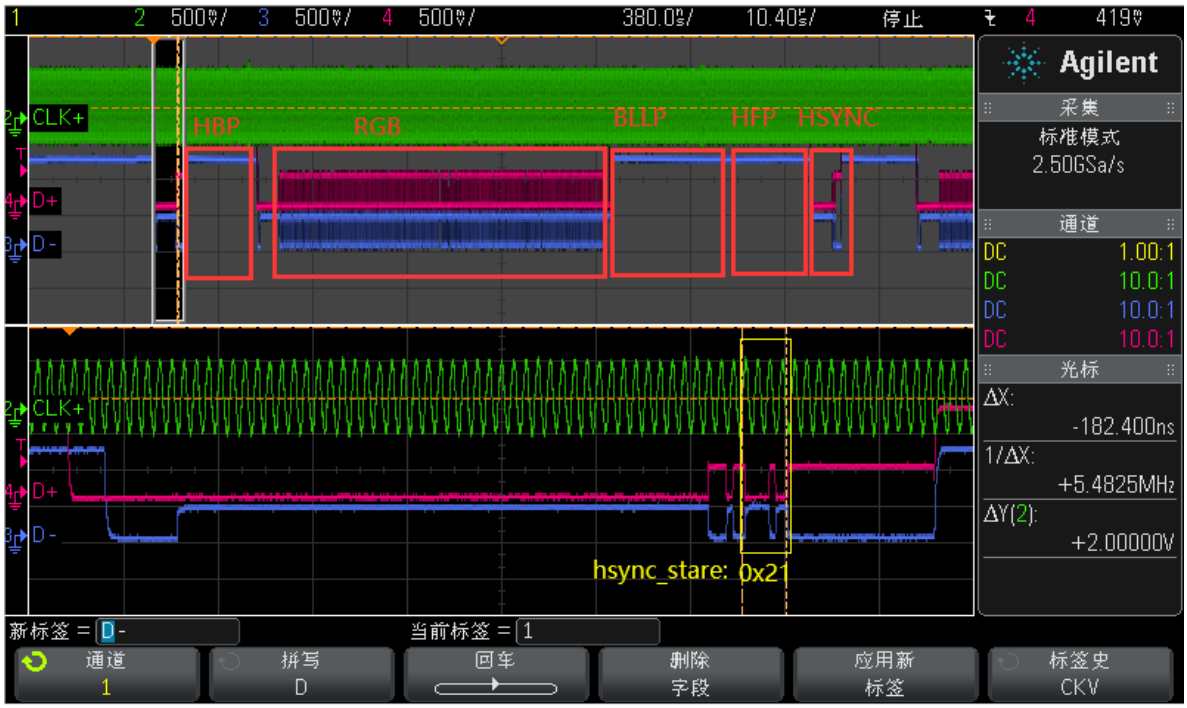
通过示波器捕获 Non Burst Sync Pulse 波形如下：

Burst Mode



Video burst mode

通过示波器捕获 Video Burst 波形如下：



常见问题

查看VOP timing 和 Connector 信息

```
console:/ # cat /d/dri/0/summary
Video Port0: DISABLED
Video Port1: DISABLED
Video Port2: DISABLED
Video Port3: ACTIVE
    Connector: DSI-2
        bus_format[100a]: RGB888_1X24
        overlay_mode[0] output_mode[0] color_space[0]
    Display mode: 1080x1920p60
        clk[132000] real_clk[132000] type[48] flag[a]
```

```
        H: 1080 1095 1099 1129
        V: 1920 1935 1937 1952
    Cluster3-win0: ACTIVE
        win_id: 6
        format: AB24 little-endian (0x34324241) SDR[0] color_space[0]
glb_alpha[0xff]
        rotate: xmirror: 0 ymirror: 0 rotate_90: 0 rotate_270: 0
        csc: y2r[0] r2y[0] csc mode[0]
        zpos: 0
        src: pos[0, 0] rect[1080 x 1920]
        dst: pos[0, 0] rect[1080 x 1920]
        buf[0]: addr: 0x0000000000376e000 pitch: 4352 offset: 0
```

查看 DSI2 相关 clk tree

```
console:/ # cat /d/clk/clk_summary | grep vop
    clk_vop_pmu                       0        0        0    24000000          0
    0  50000
          dclk_vop3                   1        2        0    33000000          0
    0  50000
          dclk_vop1_src               0        1        0   594000000          0
    0  50000
             dclk_vop1                0        1        0   594000000          0
    0  50000
          dclk_vop0_src               0        1        0   594000000          0
    0  50000
             dclk_vop0                0        1        0   594000000          0
    0  50000
             aclk_vop_low_root        1        1        0   396000000          0
    0  50000
             hclk_vop_root            2        4        0   198000000          0
    0  50000
                hclk_vop              1        3        0   198000000          0
    0  50000
          aclk_vop_root               1        1        0   500000000          0
    0  50000
             aclk_vop_doby            0        0        0   500000000          0
    0  50000
             aclk_vop_sub_src         1        1        0   500000000          0
    0  50000
                aclk_vop              1        4        0   500000000          0
    0  50000
             pclk_vop_root            3        5        0   100000000          0
    0  50000
          dclk_vop2_src               1        1        0   148500000          0
    0  50000
             dclk_vop2                1        2        0   148500000          0
    0  50000
console:/ # cat /d/clk/clk_summary | grep dsi
                pclk_dsihost1         1        2        0   100000000          0
    0  50000
                pclk_dsihost0         1        2        0   100000000          0
    0  50000
    clk_dsihost1                      1        2        0   351000000          0
    0  50000
    clk_dsihost0                      1        2        0   351000000          0
    0  50000
```

```
console:/ # cat /d/clk/clk_summary | grep mipi
          mipi1_clk_src                0        0        0    33000000          0
    0  50000
             mipi1_pixclk              0        0        0    33000000          0
    0  50000
          mipi0_clk_src                0        0        0   148500000          0
    0  50000
             mipi0_pixclk              0        0        0   148500000          0
    0  50000
    clk_usbdpphy_mipidcpphy_ref       5        5        0    24000000          0
    0  50000
    clk_mipi_camaraout_m4             0        0        0    24000000          0
    0  50000
    clk_mipi_camaraout_m3             0        0        0    24000000          0
    0  50000
    clk_mipi_camaraout_m2             0        0        0    24000000          0
    0  50000
    clk_mipi_camaraout_m0             0        0        0    24000000          0
    0  50000
          clk_mipi_camaraout_m1       0        0        0    37125000          0
    0  50000
                pclk_mipi_dcphy1       1        1        0   100000000
 0     0  50000
                pclk_mipi_dcphy0       1        1        0   100000000
 0     0  50000
console:/ #
```

如何查看指定 DSI lane 速率

DSI lane 速率的指定有两种，一种是驱动自动计算:

```
dmesg | grep dsi

[   77.369812] dw-mipi-dsi2 fde20000.dsi: [drm:dw_mipi_dsi2_encoder_enable]
final DSI-Link bandwidth: 879 x 4 Mbps
```

一种是通过如下属性手动指定:

```
&dsi0 {
    ...
    rockchip,lane-rate = <1000>;
    ...
}
```

MIPI DSI2 HOST 没有自己color bar 功能，通过如下命令可以让 VOP2 投显 color bar

根据显示路由选择对应的命令:

```
vp0:
io -4 0xfdd90c08 0x1 && io -4 0xfdd90000 0xffffffff

vp1:
io -4 0xfdd90d08 0x1 && io -4 0xfdd90000 0xffffffff

vp2:
io -4 0xfdd90e08 0x1 && io -4 0xfdd90000 0xffffffff

vp3:
io -4 0xfdd90f08 0x1 && io -4 0xfdd90000 0xffffffff
```

如何判断显示异常的时候，MIPI DSI2 和 panel 是否通信正常

uboot:

```
--- a/drivers/video/drm/rockchip_panel.c
+++ b/drivers/video/drm/rockchip_panel.c
@@ -260,6 +260,7 @@ static void panel_simple_prepare(struct rockchip_panel
*panel)
    struct rockchip_panel_priv *priv = dev_get_priv(panel->dev);
    struct mipi_dsi_device *dsi = dev_get_parent_platdata(panel->dev);
    int ret;
+   u8 mode;

    if (priv->prepared)
        return;
@@ -285,6 +286,8 @@ static void panel_simple_prepare(struct rockchip_panel
*panel)
    if (plat->delay.init)
        mdelay(plat->delay.init);

+   mipi_dsi_dcs_get_power_mode(dsi, &mode);
+   printf("===>mode: 0x%x\n", mode);
    if (plat->on_cmds) {
        if (priv->cmd_type == CMD_TYPE_SPI)
            ret = rockchip_panel_send_spi_cmds(panel->state,
@@ -298,6 +301,8 @@ static void panel_simple_prepare(struct rockchip_panel
*panel)
            printf("failed to send on cmds: %d\n", ret);
    }

+   mipi_dsi_dcs_get_power_mode(dsi, &mode);
+   printf("===>mode: 0x%x\n", mode);
    priv->prepared = true;
 }
```

kernel

```
--- a/drivers/gpu/drm/panel/panel-simple.c
+++ b/drivers/gpu/drm/panel/panel-simple.c
@@ -506,6 +506,7 @@ static int panel_simple_prepare(struct drm_panel *panel)
    unsigned int delay;
    int err;
    int hpd_asserted;
+   u8 mode;
```

```
    if (p->prepared)
        return 0;
@@ -554,6 +555,8 @@ static int panel_simple_prepare(struct drm_panel *panel)
        }
    }

+   mipi_dsi_dcs_get_power_mode(p->dsi, &mode);
+   printk("===>mode: 0x%x\n, mode");
    if (p->desc->init_seq)
        if (p->dsi)
            panel_simple_xfer_dsi_cmd_seq(p, p->desc->init_seq);
@@ -561,6 +564,9 @@ static int panel_simple_prepare(struct drm_panel *panel)
    if (p->desc->delay.init)
        msleep(p->desc->delay.init);

+   mipi_dsi_dcs_get_power_mode(p->dsi, &mode);
+   printk("===>mode: 0x%x\n, mode");
+
    p->prepared = true;

    return 0;
```

通信正常会有如下打印,否者排查屏端时序确保屏是否就绪：

```
===> mode: 0x8
===> mode: 0x9c
```

backlight驱动probe失败

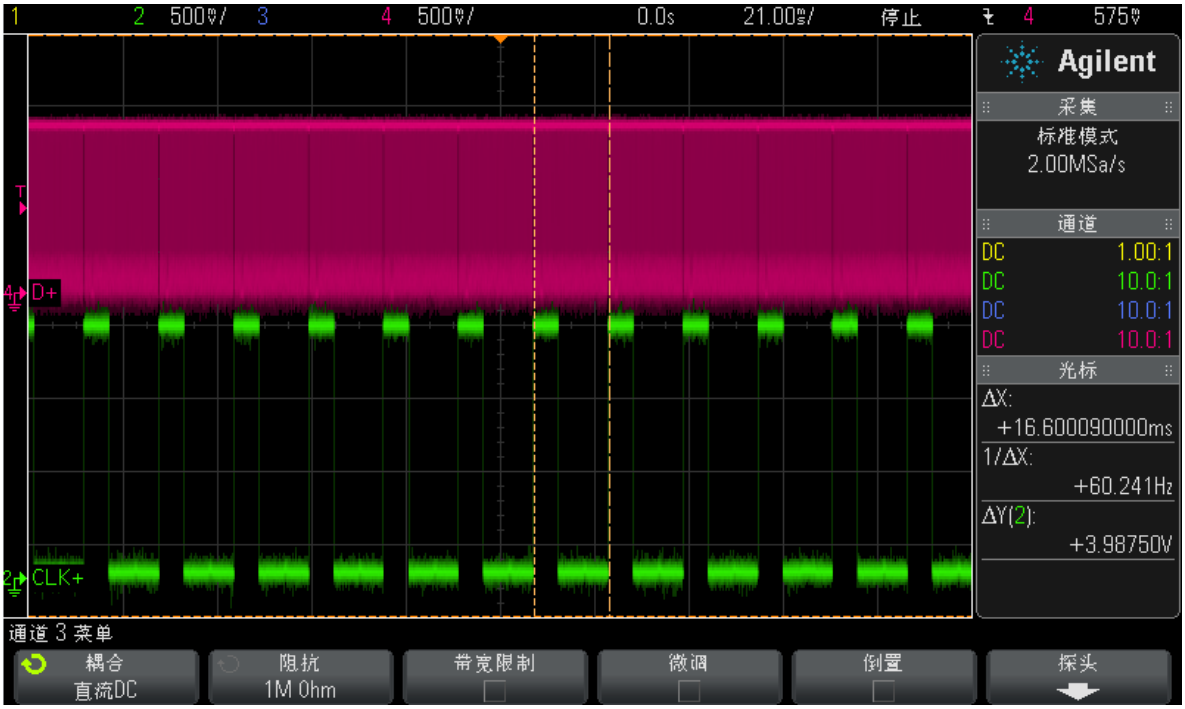
```
console:/ # dmesg | grep backlight
[    3.164274] pwm-backlight: probe of backlight failed with error -16
```

Command Mode 显示模组如何配置 TE

为防止图像显示撕裂，显示控制器刷帧的频率应该和显示模组从GRAM中刷图的频率保持一致。RK3588只支持显示模组将TE信号外部反馈的方式。

```
&dsi0 {
    ...
    /* 显示模组TE信号连接到MIPI_TE0 */
    pinctrl-names = "default";
    pinctrl-0 = <&mipi_te0>;
    ...
};

&dsi1 {
    ...
    /* 显示模组TE信号连接到MIPI_TE1 */
    pinctrl-names = "default";
    pinctrl-0 = <&mipi_te1>;
    ...
};
```

下图捕获MIPI DSI发送数据信号场频和显示模组TE信号频率一致波形：



双通道MIPI切换主从顺序

如果硬件设计将双通道的两个MIPI Ports接反了，可以通过如下配置进行软件纠正：

```
&dsi0 {
    ...
    rockchip,data-swap;
    rockchip,dual-channel = <&dsi1>;
    ...
};

&dsi1 {
    status = "okay";
};
```

调试节点

在MIPI DSI信号测试过程中需要对信号进行调整，如下是相关调试节点路径：

```
dcphy0：
cd /sys/devices/platform/feda0000.phy/
dcphy1：
cd /sys/devices/platform/fedb0000.phy/
```

Patch

```
diff --git a/drivers/phy/rockchip/phy-rockchip-samsung-dcphy.c
b/drivers/phy/rockchip/phy-rockchip-samsung-dcphy.c
index 1d5db69ee..c5d11f30c 100644
--- a/drivers/phy/rockchip/phy-rockchip-samsung-dcphy.c
+++ b/drivers/phy/rockchip/phy-rockchip-samsung-dcphy.c
@@ -142,6 +142,25 @@
 #define T_TA_GET(x)         UPDATE(x, 7, 4)
 #define T_TA_GO(x)      UPDATE(x, 3, 0)
```

```
+
+#define REG_400M_MASK      GENMASK(6, 4)
+#define REG_400M(x)        UPDATE(x, 6, 4)
+#define BIAS_CON4      0x0010
+#define I_MUX_SEL_MASK     GENMASK(6, 5)
+#define I_MUX_SEL(x)       UPDATE(x, 6, 5)
+#define CAP_PEAKING_MASK   GENMASK(14, 12)
+#define CAP_PEAKING(x)     UPDATE(x, 14, 12)
+#define RES_UP_MASK        GENMASK(7, 4)
+#define RES_UP(x)      UPDATE(x, 7, 4)
+#define RES_DN_MASK        GENMASK(3, 0)
+#define RES_DN(x)      UPDATE(x, 3, 0)
+#define T_HS_ZERO_MASK     GENMASK(15, 8)
+#define T_HS_PREPARE_MASK  GENMASK(7, 0)
+#define T_HS_EXIT_MASK     GENMASK(15, 8)
+#define T_HS_TRAIL_MASK        GENMASK(7, 0)
+#define T_TA_GET_MASK      GENMASK(7, 4)
+#define T_TA_GO_MASK       GENMASK(3, 0)
+
 /* MIPI_CDPHY_GRF registers */
 #define MIPI_DCPHY_GRF_CON0    0x0000
 #define S_CPHY_MODE        HIWORD_UPDATE(1, 3, 3)
@@ -1194,6 +1213,421 @@ struct samsung_mipi_cphy_timing
samsung_mipi_cphy_timing_table[] = {
    {   80,  1, 50, 25,  2,  0, 2 },
 };

+static ssize_t
+reg_400m_show(struct device *device, struct device_attribute *attr, char *buf)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned int val;
+   unsigned int level;
+
+   regmap_read(samsung->regmap, BIAS_CON2, &val);
+   level = (val & REG_400M_MASK) >> 4;
+
+   return sprintf(buf, "%d\n", level);
+
+}
+
+static ssize_t
+reg_400m_store(struct device *device, struct device_attribute *attr,
+           const char *buf, size_t count)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned long val;
+
+   if (kstrtoul(buf, 10, &val))
+       return -EINVAL;
+
+
+   if (val > 7)
+       val = 7;
+
+   regmap_update_bits(samsung->regmap, BIAS_CON2, REG_400M_MASK,
REG_400M(val));
+
```

```
+	return count;
+}
+static DEVICE_ATTR_RW(reg_400m);
+
+static ssize_t
+cap_peaking_show(struct device *device, struct device_attribute *attr, char
*buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int level;
+
+	regmap_read(samsung->regmap, COMBO_MD0_ANA_CON0, &val);
+	level = (val & CAP_PEAKING_MASK) >> 12;
+
+	return sprintf(buf, "%d\n", level);
+
+}
+
+static ssize_t
+cap_peaking_store(struct device *device, struct device_attribute *attr,
+			const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+
+	if (val > 7)
+		val = 7;
+
+	regmap_update_bits(samsung->regmap, DPHY_MC_ANA_CON0, CAP_PEAKING_MASK,
CAP_PEAKING(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD0_ANA_CON0, CAP_PEAKING_MASK,
CAP_PEAKING(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD1_ANA_CON0, CAP_PEAKING_MASK,
CAP_PEAKING(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD2_ANA_CON0, CAP_PEAKING_MASK,
CAP_PEAKING(val));
+	regmap_update_bits(samsung->regmap, DPHY_MD3_ANA_CON0, CAP_PEAKING_MASK,
CAP_PEAKING(val));
+
+	return count;
+}
+static DEVICE_ATTR_RW(cap_peaking);
+
+static ssize_t
+res_up_show(struct device *device, struct device_attribute *attr, char *buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int level;
+
+	regmap_read(samsung->regmap, COMBO_MD0_ANA_CON0, &val);
+	level = (val & RES_UP_MASK) >> 4;
+
```

```
+	return sprintf(buf, "%d\n", level);
+
+}
+
+static ssize_t
+res_up_store(struct device *device, struct device_attribute *attr,
+	     const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+
+	if (val > 15)
+		val = 15;
+
+	regmap_update_bits(samsung->regmap, DPHY_MC_ANA_CON0, RES_UP_MASK,
RES_UP(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD0_ANA_CON0, RES_UP_MASK,
RES_UP(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD1_ANA_CON0, RES_UP_MASK,
RES_UP(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD2_ANA_CON0, RES_UP_MASK,
RES_UP(val));
+	regmap_update_bits(samsung->regmap, DPHY_MD3_ANA_CON0, RES_UP_MASK,
RES_UP(val));
+
+	return count;
+}
+static DEVICE_ATTR_RW(res_up);
+
+static ssize_t
+res_dn_show(struct device *device, struct device_attribute *attr, char *buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int level;
+
+	regmap_read(samsung->regmap, COMBO_MD0_ANA_CON0, &val);
+	level = (val & RES_DN_MASK);
+
+	return sprintf(buf, "%d\n", level);
+
+}
+
+static ssize_t
+res_dn_store(struct device *device, struct device_attribute *attr,
+	     const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+
```

```
+	if (val > 15)
+		val = 15;
+
+	regmap_update_bits(samsung->regmap, DPHY_MC_ANA_CON0, RES_DN_MASK,
RES_DN(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD0_ANA_CON0, RES_DN_MASK,
RES_DN(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD1_ANA_CON0, RES_DN_MASK,
RES_DN(val));
+	regmap_update_bits(samsung->regmap, COMBO_MD2_ANA_CON0, RES_DN_MASK,
RES_DN(val));
+	regmap_update_bits(samsung->regmap, DPHY_MD3_ANA_CON0, RES_DN_MASK,
RES_DN(val));
+
+	return count;
+}
+static DEVICE_ATTR_RW(res_dn);
+
+static ssize_t
+output_voltage_show(struct device *device, struct device_attribute *attr, char
*buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int level;
+
+	regmap_read(samsung->regmap, BIAS_CON4, &val);
+	level = (val & I_MUX_SEL_MASK) >> 5;
+
+	return sprintf(buf, "%d\n", level);
+
+}
+
+static ssize_t
+output_voltage_store(struct device *device, struct device_attribute *attr,
+			const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+
+	if (val > 3)
+		val = 3;
+
+	regmap_update_bits(samsung->regmap, BIAS_CON4,
+			   I_MUX_SEL_MASK, I_MUX_SEL(val));
+
+	return count;
+}
+static DEVICE_ATTR_RW(output_voltage);
+
+static ssize_t
+hs_exit_show(struct device *device, struct device_attribute *attr, char *buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
```

```
+	unsigned int val;
+	unsigned int hs_exit;
+
+	regmap_read(samsung->regmap, COMBO_MD0_TIME_CON2, &val);
+	hs_exit = (val & GENMASK(15, 8)) >> 8;
+
+	return sprintf(buf, "%d\n", hs_exit);
+
+}
+
+static ssize_t hs_exit_store(struct device *device, struct device_attribute
*attr,
+			const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+	unsigned long hs_exit;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+	hs_exit = T_HS_EXIT(val);
+	regmap_update_bits(samsung->regmap, COMBO_MD0_TIME_CON2, T_HS_EXIT_MASK,
hs_exit);
+	regmap_update_bits(samsung->regmap, COMBO_MD1_TIME_CON2, T_HS_EXIT_MASK,
hs_exit);
+	regmap_update_bits(samsung->regmap, COMBO_MD2_TIME_CON2, T_HS_EXIT_MASK,
hs_exit);
+
+	if (!samsung->c_option) {
+		regmap_update_bits(samsung->regmap, DPHY_MC_TIME_CON2, T_HS_EXIT_MASK,
hs_exit);
+		regmap_update_bits(samsung->regmap, DPHY_MD3_TIME_CON2, T_HS_EXIT_MASK,
hs_exit);
+	}
+
+	return count;
+}
+static DEVICE_ATTR_RW(hs_exit);
+
+static ssize_t
+hs_trail_or_post_3_show(struct device *device, struct device_attribute *attr,
char *buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int hs_trail;
+
+	regmap_read(samsung->regmap, COMBO_MD0_TIME_CON2, &val);
+	hs_trail = val & GENMASK(7, 0);
+
+	return sprintf(buf, "%d\n", hs_trail);
+
+}
+
+static ssize_t hs_trail_or_post_3_store(struct device *device, struct
device_attribute *attr,
+			const char *buf, size_t count)
```

```
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned long val;
+   unsigned long hs_trail;
+
+   if (kstrtoul(buf, 10, &val))
+       return -EINVAL;
+
+   hs_trail = T_HS_TRAIL(val);
+   regmap_update_bits(samsung->regmap, COMBO_MD0_TIME_CON2, T_HS_TRAIL_MASK,
hs_trail);
+   regmap_update_bits(samsung->regmap, COMBO_MD1_TIME_CON2, T_HS_TRAIL_MASK,
hs_trail);
+   regmap_update_bits(samsung->regmap, COMBO_MD2_TIME_CON2, T_HS_TRAIL_MASK,
hs_trail);
+
+   if (!samsung->c_option) {
+       regmap_update_bits(samsung->regmap, DPHY_MC_TIME_CON2, T_HS_TRAIL_MASK,
hs_trail);
+       regmap_update_bits(samsung->regmap, DPHY_MD3_TIME_CON2, T_HS_TRAIL_MASK,
hs_trail);
+   }
+
+   return count;
+}
+static DEVICE_ATTR_RW(hs_trail_or_post_3);
+
+static ssize_t
+hs_zero_or_prebegin_3_show(struct device *device, struct device_attribute
*attr, char *buf)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned int val;
+   unsigned int hs_zero;
+
+   regmap_read(samsung->regmap, COMBO_MD0_TIME_CON1, &val);
+   hs_zero = (val & GENMASK(15, 8)) >> 8;
+
+   return sprintf(buf, "%d\n", hs_zero);
+
+}
+
+static ssize_t hs_zero_or_prebegin_3_store(struct device *device, struct
device_attribute *attr,
+            const char *buf, size_t count)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned long val;
+   unsigned long hs_zero;
+
+   if (kstrtoul(buf, 10, &val))
+       return -EINVAL;
+
+   hs_zero = T_HS_ZERO(val);
+   regmap_update_bits(samsung->regmap, COMBO_MD0_TIME_CON1,
+              T_HS_ZERO_MASK, hs_zero);
+   regmap_update_bits(samsung->regmap, COMBO_MD1_TIME_CON1,
+              T_HS_ZERO_MASK, hs_zero);
```

```
+	regmap_update_bits(samsung->regmap, COMBO_MD2_TIME_CON1,
+			T_HS_ZERO_MASK, hs_zero);
+
+	if (!samsung->c_option) {
+		regmap_update_bits(samsung->regmap, DPHY_MC_TIME_CON1,
+				T_HS_ZERO_MASK, hs_zero);
+		regmap_update_bits(samsung->regmap, DPHY_MD3_TIME_CON1,
+				T_HS_ZERO_MASK, hs_zero);
+	}
+
+	return count;
+}
+static DEVICE_ATTR_RW(hs_zero_or_prebegin_3);
+
+static ssize_t
+hs_prepare_or_prepare_3_show(struct device *device, struct device_attribute
*attr, char *buf)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned int val;
+	unsigned int hs_prepare;
+
+	regmap_read(samsung->regmap, COMBO_MD0_TIME_CON1, &val);
+	hs_prepare = val & GENMASK(7, 0);
+
+	return sprintf(buf, "%d\n", hs_prepare);
+}
+
+static ssize_t hs_prepare_or_prepare_3_store(struct device *device, struct
device_attribute *attr,
+			const char *buf, size_t count)
+{
+	struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+	unsigned long val;
+	unsigned long hs_prepare;
+
+	if (kstrtoul(buf, 10, &val))
+		return -EINVAL;
+
+	hs_prepare = T_HS_PREPARE(val);
+	regmap_update_bits(samsung->regmap, COMBO_MD0_TIME_CON1,
+			T_HS_PREPARE_MASK, hs_prepare);
+	regmap_update_bits(samsung->regmap, COMBO_MD1_TIME_CON1,
+			T_HS_PREPARE_MASK, hs_prepare);
+	regmap_update_bits(samsung->regmap, COMBO_MD2_TIME_CON1,
+			T_HS_PREPARE_MASK, hs_prepare);
+
+	if (!samsung->c_option) {
+		regmap_update_bits(samsung->regmap, DPHY_MC_TIME_CON1,
+				T_HS_PREPARE_MASK, hs_prepare);
+		regmap_update_bits(samsung->regmap, DPHY_MD3_TIME_CON1,
+				T_HS_PREPARE_MASK, hs_prepare);
+	}
+
+	return count;
+}
+static DEVICE_ATTR_RW(hs_prepare_or_prepare_3);
+
```

```
+static ssize_t
+lpx_show(struct device *device, struct device_attribute *attr, char *buf)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned int val;
+   unsigned int lpx;
+
+   regmap_read(samsung->regmap, COMBO_MD0_TIME_CON0, &val);
+   lpx = (val & GENMASK(11, 4)) >> 4;
+
+   return sprintf(buf, "%d\n", lpx);
+
+}
+
+static ssize_t lpx_store(struct device *device, struct device_attribute *attr,
+            const char *buf, size_t count)
+{
+   struct samsung_mipi_dcphy *samsung = dev_get_drvdata(device);
+   unsigned long val;
+   unsigned long lpx = 0;
+
+   if (kstrtoul(buf, 10, &val))
+       return -EINVAL;
+
+   lpx |= T_LPX(val);
+   /*  T_LP_EXIT_SKEW/T_LP_ENTRY_SKEW unconfig */
+   regmap_write(samsung->regmap, COMBO_MD0_TIME_CON0, lpx);
+   regmap_write(samsung->regmap, COMBO_MD1_TIME_CON0, lpx);
+   regmap_write(samsung->regmap, COMBO_MD2_TIME_CON0, lpx);
+
+   if (!samsung->c_option) {
+       regmap_write(samsung->regmap, DPHY_MC_TIME_CON0, lpx);
+       regmap_write(samsung->regmap, DPHY_MD3_TIME_CON0, lpx);
+   }
+
+   return count;
+}
+static DEVICE_ATTR_RW(lpx);
+
+static struct attribute *samsung_mipi_dcphy_cts_attrs[] = {
+   &dev_attr_lpx.attr,
+   &dev_attr_hs_prepare_or_prepare_3.attr,
+   &dev_attr_hs_zero_or_prebegin_3.attr,
+   &dev_attr_hs_trail_or_post_3.attr,
+   &dev_attr_hs_exit.attr,
+   &dev_attr_output_voltage.attr,
+   &dev_attr_res_up.attr,
+   &dev_attr_res_dn.attr,
+   &dev_attr_cap_peaking.attr,
+   &dev_attr_reg_400m.attr,
+   NULL
+};
+
+static const struct attribute_group samsung_mipi_dcphy_cts_attr_group = {
+   .attrs = samsung_mipi_dcphy_cts_attrs,
+};
+
+static int samsung_mipi_dcphy_cts_sysfs_add(struct samsung_mipi_dcphy *samsung)
```

```
+{
+   struct device *dev = samsung->dev;
+   int ret;
+
+   ret = sysfs_create_group(&dev->kobj, &samsung_mipi_dcphy_cts_attr_group);
+   if (ret) {
+       dev_err(dev, "failed to register sysfs. err: %d\n", ret);
+       return ret;
+   };
+
+   return 0;
+}
+
 static void samsung_mipi_dcphy_bias_block_enable(struct samsung_mipi_dcphy
*samsung)
 {
    regmap_write(samsung->regmap, BIAS_CON0, 0x0010);
@@ -1912,6 +2346,11 @@ static int samsung_mipi_dcphy_probe(struct
platform_device *pdev)
        return PTR_ERR(phy_provider);
    }

+
+   ret = samsung_mipi_dcphy_cts_sysfs_add(samsung);
+   if (ret)
+       return ret;
+
    pm_runtime_enable(dev);

    return 0;
```

驱动强度

```
echo level > output_voltage
```

驱动默认如下配置：

1. D-PHY: 2'b00
2. C-PHY: 2'b10

level 参考如下：

1. 2b'00 : 400mV
2. 2b'01 : 200mV
3. 2b'10 : 530mV
4. 2b'11 : 530mV

共模电压

```
echo level > reg_400m
```

level 参考如下：

1. 3'b000: 380mV / 230mV
2. 3'b001: 390mV / 220mV

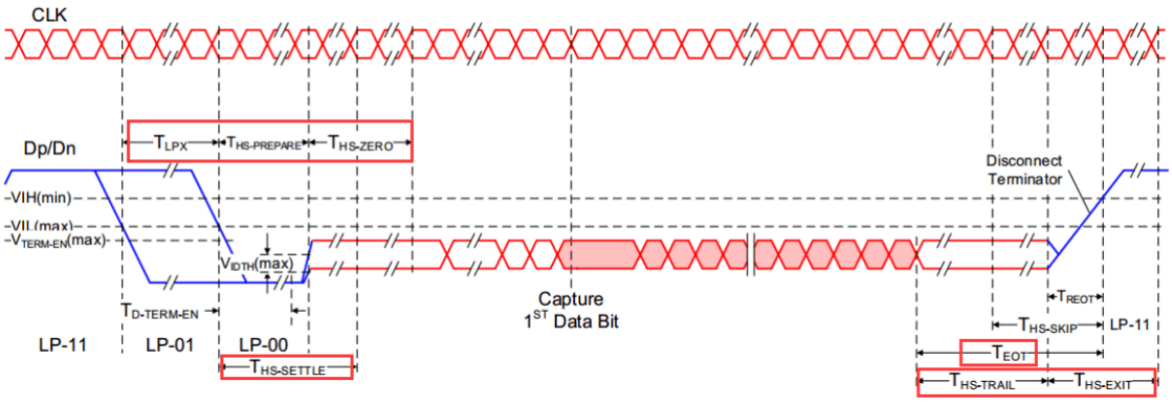
3. 3'b010: 400mV / 210mV
4. 3'b011: 410mV / 200mV
5. 3'b100: 420mV / 190mV
6. 3'b101: 430mV / 180mV
7. 3'b110: 440mV / 170mV
8. 3'b111: 450mV / 160mV

cap peaking

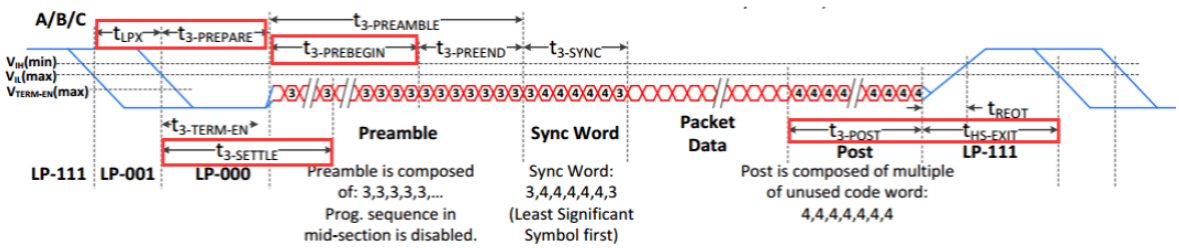
```
echo level > cap_peaking

level: 0~7
```

信号timing

如下图中信号红色框中的信号参数都是可以调整的。



D-PHY High-Speed Data Transmission in Burst



C-PHY High-Speed Data Transmission in Burst

如下信号调整count值可以在回读基础上做调整修改，比如Tlpx:

```
cat lpx
```

Tlpx

```
echo count > lpx
```

Ths_prepare_or_prepare_3

```
echo count > hs_prepare_or_prepare_3
```

Ths_zero_or_prebegin_3

```
echo count > hs_zero_or_prebegin_3
```

Ths_trail_or_post_3

```
echo count > hs_trail_or_post_3
```

Ths_exit

```
echo count > hs_exit
```

High-Speed Driver Up Resistor Control

```
echo level > res_up
```

level参考如下：

1. 4'b0000: 43 ohm
2. 4'b0001: 46 ohm
3. 4'b0010: 49 ohm
4. 4'b0011: 52 ohm
5. 4'b0100: 56 ohm
6. 4'b0101: 60 ohm
7. 4'b0110: 66 ohm
8. 4'b0111: 73 ohm
9. 4'b1000: 30 ohm
10. 4'b1001: 31.2 ohm
11. 4'b1010: 32.5 ohm
12. 4'b1011: 34 ohm
13. 4'b1100: 35.5 ohm
14. 4'b1101: 37 ohm
15. 4'b1110: 39 ohm
16. 4'b1111: 41 ohm

High-Speed Driver Down Resistor Control

```
echo level > res_dn
```

level 参考如下：

1. 4'b0000: 43 ohm
2. 4'b0001: 46 ohm
3. 4'b0010: 49 ohm
4. 4'b0011: 52 ohm
5. 4'b0100: 56 ohm
6. 4'b0101: 60 ohm
7. 4'b0110: 66 ohm
8. 4'b0111: 73 ohm
9. 4'b1000: 30 ohm
10. 4'b1001: 31.2 ohm
11. 4'b1010: 32.5 ohm
12. 4'b1011: 34 ohm
13. 4'b1100: 35.5 ohm
14. 4'b1101: 37 ohm
15. 4'b1110: 39 ohm
16. 4'b1111: 41 ohm