HDMI2.0 Transmitter

上海安路信息科技股份有限公司

APUG_093（v1.0）2024 年 9 月

目 录

1 概述

HDMI(High Definition Multimedia Interface)是一种全数字化影像和声音收发接口，可以收发未压缩的音频及视频信号。HDMI 可用于机顶盒、DVD 播放机、个人电脑、投影仪与电视等设备，HDMI 收发音频和视频信号采用同一条线材，大大简化系统线路的安装难度。

本文档描述了 HDMI2.0 Transmitter 设计，符合 HDMI2.0/1.4b 协议标准，实现音频和视频数据的高速发送，并在 AP106_V2.0 开发板上进行开发验证。

本 HDMI2.0 Transmitter 设计支持的具体功能如下：

- 符合 HDMI2.0 和 HDMI1.4b 协议标准。
- 最高支持 4Kp60 显示分辨率。
- 支持 8 路音频。
- 支持 1 像素、2 像素和 4 像素模式。
- 支持 24bit、30bit 色深。
- 支持 RGB、YUV444 和 YUV422 视频格式。
- 支持内部视频测试源。
- 支持 DDC 通道。
- 支持分辨率动态切换。

2 功能描述

2.1 设计框架

HDMI2.0 Transmitter 设计框架如图 2-1 所示。

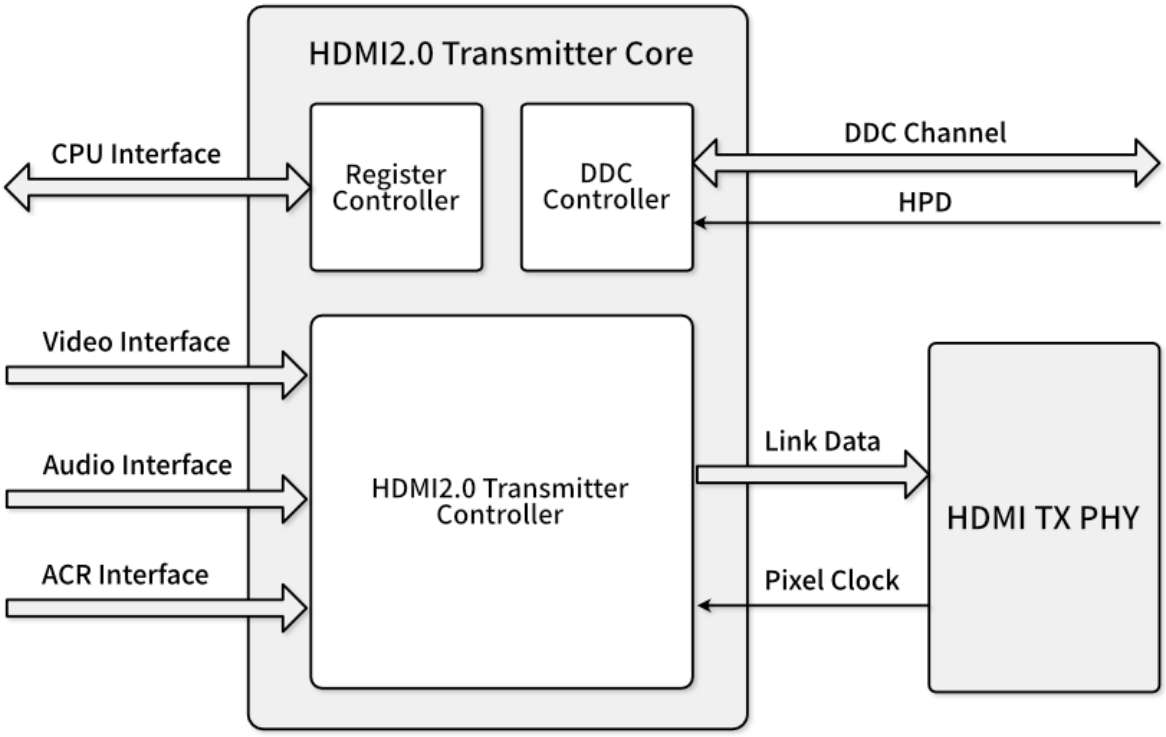

图 2-1 HDMI2.0 Transmitter 设计框架

HDMI2.0 Transmitter 系统框架包含协议层 HDMI2.0 Transmitter Core 和物理层 HDMI TX PHY。其中 HDMI2.0 Transmitter Core 接口包括 CPU Interface、Video Interface、Audio Interface、ACR Interface、DDC Channel、HPD、Link Data。

HDMI TX PHY 采用高速 SerDes 实现，以实现高达 4K60 分辨率的发送。

2.2 时钟架构

在使用 HDMI2.0 Transmitter 时，时钟的配置至关重要，用户逻辑和 HDMI2.0 Transmitter 的结构如图 2-2 所示。

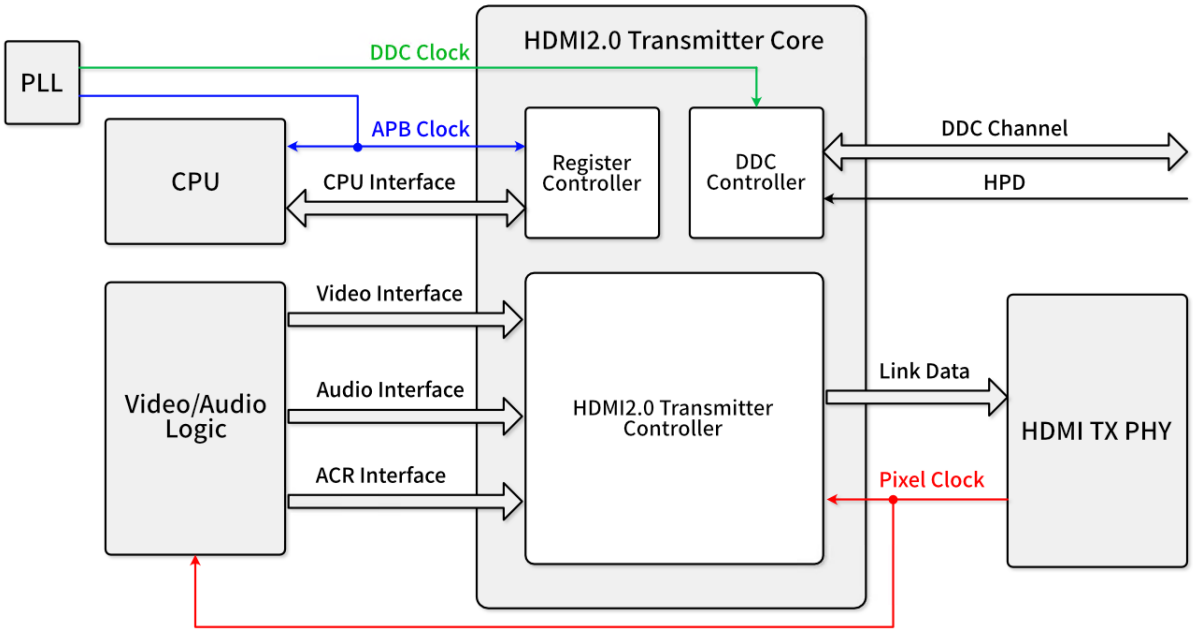

图 2-2 HDMI2.0 Transmitter 时钟架构

HDMI2.0 Transmitter Core 有三个时钟， APB Clock、DDC Clock 和 Pixel Clock。

APB Clock 是 CPU 总线时钟，配置总线和内部的控制寄存器都工作在此时钟域下。

DDC Clock 是 DDC 控制逻辑时钟，DDC 相关的功能都是工作在此时钟域下。

Pixel Clock 是 HDMI2.0 Transmitter 控制器时钟，HDMI 视频和音频数据的处理和编码都是工作在此时钟域下，一般来源于 SerDes 的用户时钟。

2.3 接口及参数

2.3.1 参数说明

HDMI2.0 Transmitter Core 参数说明如表 2-1 所示。

表 2-1 HDMI2.0 Transmitter Core 参数

参数	可选值	默认值	说明
DEVICE	"PH1A","PH1P","PH2A","DR1"	"PH1A"	器件类型选择
DDC_SCL_DIV	1-65535	125	DDC 通道中 IIC 时钟分频系数

DEVICE

DEVICE 用来设定器件类型，此参数决定了内部原语的器件类型，需要根据所使用的器件来设定，否则会引起综合报错。

DDC_SCL_DIV

DDC_SCL_DIV 用来设定 DDC 通道中 IIC 时钟的分频系数，DDC IIC 的时钟频率计算如下：

$$f_{ddc\_scl} = f_{ddc\_clk}/(2 * DDC\_SCL\_DIV)$$

其中f_{ddc_scl}为 DDC 通道中 IIC 的时钟频率，f_{ddc_clk}是 DDC Clock 的时钟频率，如果 DDC Clock 的时钟频率为 50MHz，DDC_SCL_DIV 为 125，那么 DDC 的通信频率为 50MHz/(2*125)=200KHz。

2.3.2 接口信号说明

HDMI2.0 Transmitter Core 的接口如图 2-3 所示。

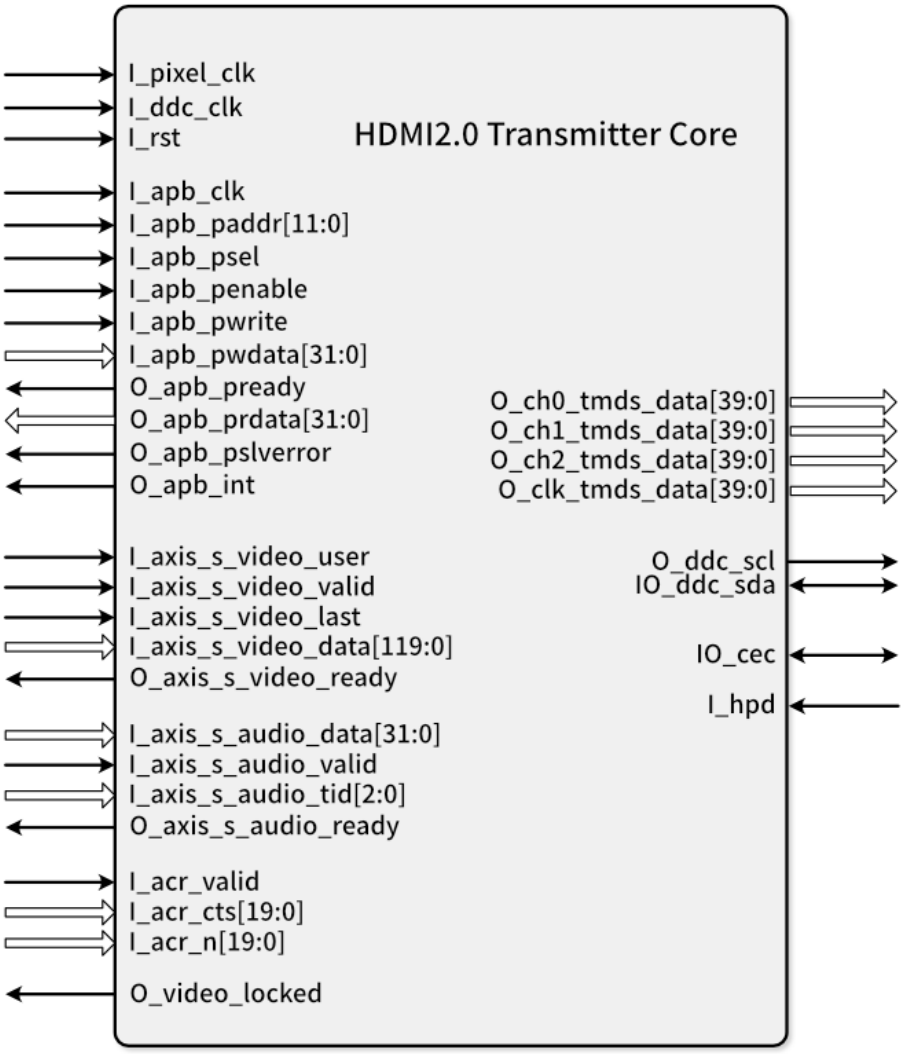

图 2-3 HDMI2.0 Transmitter Core 接口

HDMI2.0 Transmitter Core 接口包括工作所需要的时钟和复位信号，CPU 配置总线，视频数据输入接口，音频数据输入接口，音频时钟再生辅助数据包 Audio Clock Regeneration(ACR)输入接口，TMDS 编码输出接口，DDC 信号和热插拔信号。

接口说明如表 2-2 所示。

表 2-2 HDMI2.0 Transmitter Core 接口信号

接口名称	信号名称	方向	时钟域	描述
Clock And Reset	I_pixel_clk	in	NA	像素时钟
	I_ddc_clk	in	NA	DDC 通道处理时钟
	I_rst	in	NA	异步复位信号，高电平有效
CPU Interface	I_apb_clk	in	NA	总线时钟

	I_apb_paddr[11:0]	in	I_apb_clk	总线地址
	I_apb_psel	in	I_apb_clk	总线选择
	I_apb_penable	in	I_apb_clk	总线使能
	I_apb_pwrite	in	I_apb_clk	总线方向
	I_apb_pwdata[31:0]	in	I_apb_clk	总线写数据
	O_apb_pready	out	I_apb_clk	总线准备信号
	O_apb_prdata[31:0]	out	I_apb_clk	总线读数据
	O_apb_pslverror	out	I_apb_clk	保留，输出 0
	O_apb_int	out	I_apb_clk	保留，输出 0
Video Interface	I_axis_s_video_user	in	I_pixel_clk	视频帧起始信号
	I_axis_s_video_valid	in	I_pixel_clk	视频有效信号
	I_axis_s_video_last	in	I_pixel_clk	视频行结束信号
	I_axis_s_video_data[119:0]	in	I_pixel_clk	视频数据
	O_axis_s_video_ready	out	I_pixel_clk	视频 ready 信号
Audio Interface	I_axis_s_audio_valid	in	I_pixel_clk	音频有效信号
	I_axis_s_audio_tid[2:0]	in	I_pixel_clk	音频通道
	I_axis_s_audio_data[31:0]	in	I_pixel_clk	音频数据 [31:28]:Preamble Code 4'b0001:start of block 4'b0000:otherwise [27] :P(Parity Bit) [26] :C(Channel Status Bit) [25] :U(User Bit) [24] :V(Validity Bit) [23:0] :Audio Sample Data
	O_axis_s_audio_ready	out	I_pixel_clk	音频 ready 信号
ACR Interface	I_acr_valid	in	I_pixel_clk	ACR 有效信号
	I_acr_cts[19:0]	in	I_pixel_clk	ACR CTS
	I_acr_n[19:0]	in	I_pixel_clk	ACR N
Status Signal	O_video_locked	out	I_pixel_clk	输入视频接口锁定信号
Link Data	O_ch0_tmds_data[39:0]	out	I_pixel_clk	通道 0 编码数据输出
	O_ch1_tmds_data[39:0]	out	I_pixel_clk	通道 1 编码数据输出
	O_ch2_tmds_data[39:0]	out	I_pixel_clk	通道 2 编码数据输出
	O_clk_tmds_data[39:0]	out	I_pixel_clk	时钟通道编码数据输出
HDMI DDC Interface	O_ddc_scl	out	I_ddc_clk	DDC 通道时钟
	IO_ddc_sda	inout	I_ddc_clk	DDC 通道数据
HDMI CEC Signal	IO_cec	inout	NA	保留
HDMI HPD Signal	I_hpd	in	NA	热插拔信号

2.4 接口时序

2.4.1 CPU Interface

CPU Interface 采用 APB 总线实现， 用于配置 HDMI2.0 Transmitter Core 内部的所有控制寄存器，同时 DDC 通道中 EDID 和 SCDC 的读写以及 HPD 状态的读取都是通过 APB 总线完成。

APB 总线写时序如图 2-4 所示。

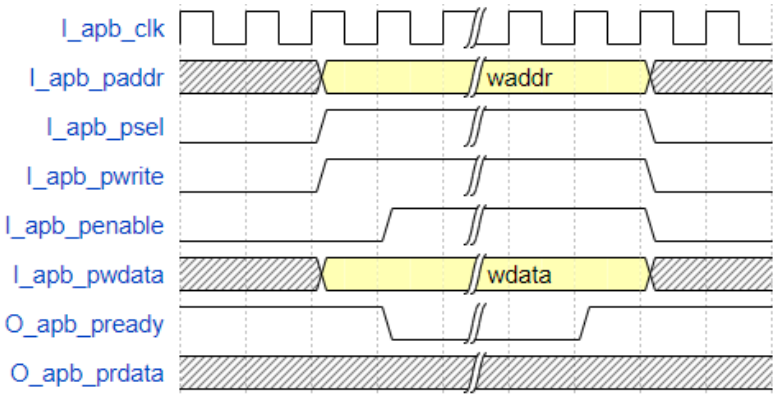

图 2-4 APB 总线写时序

APB 总线读时序如图 2-5 所示。

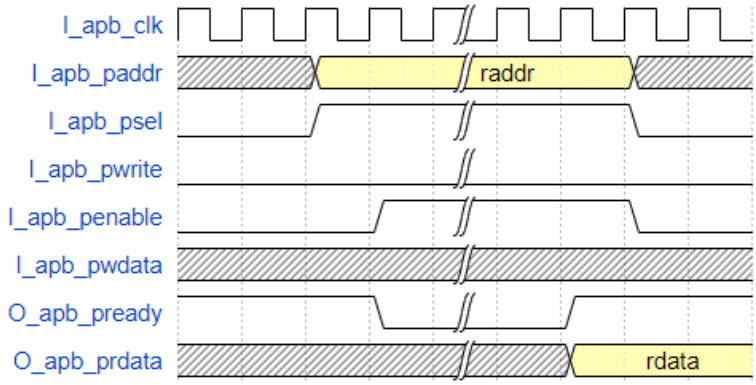

图 2-5 APB 总线读时序

HDMI2.0 Transmitter Core 的寄存器列表见表 2-3 所示，需要注意的是：**APB 总线固定是 8bit 地址，也就是 CPU 通过 APB 总线访问内部寄存器时，地址需要定义为 uint8_t 类型，否则会导致寄存器读写出错。**

表 2-3 HDMI2.0 Transmitter Core 寄存器列表

地址	读写类型	寄存器名称	字节数	描述
000h~0FFh	rd only	hdmi edid data	256 bytes	存放从 Sink 端读取的 EDID 数据
100h~1FFh	rd and wr	hdmi scdc data	256 bytes	Sink 端 SCDC 数据
200h~21Eh	rd and wr	General Control	31 bytes	HDMI 辅助数据包配置地址，前 3 个字节是 Packet Header，后 28 个字节是 Packet Body。Header 和 Body 的校验数据在内部会自动计算并填入。
21Fh~23Dh	rd and wr	Vendor-Specific InfoFrame	31 bytes	
23Eh~25Ch	rd and wr	AVI InfoFrame	31 bytes	
25Dh~27Bh	rd and wr	Source Product Description InfoFrame	31 bytes	
27Ch~29Ah	rd and wr	Audio InfoFrame	31 bytes	
29Bh~2B9h	rd and wr	MPEG Source InfoFrame	31 bytes	
2C0h~2DEh	rd and wr	any island packet	31 bytes	
300h	rd and wr	controller_rst	1 byte	HDMI 控制器复位 0x00：复位释放 0x01：复位有效
301h	rd and wr	all_island_packet_send_en	1 byte	所有辅助数据包发送使能 0x00：禁止辅助数据包发送 0x01：使能辅助数据包发送
302h	rd and wr	soft_any_packet_send_trig	1 byte	软件任意辅助数据包发送触发，上升沿有效 0x00：清零 0x01：触发一次任意辅助数据包发送，下次发送需要先清 0 再置 1
303h	rd and wr	edid_read_trig	1 byte	EDID 读取触发，上升沿有效 0x00：清零 0x01：触发一次 EDID 读操作，下次读取需要先清 0 再置 1
304h	rd only	edid_read_done	1 byte	EDID 读取完成标志位，为高之后才能去 EDID 缓存地址中读取 EDID 数据，在 edid_read_trig 拉高之后清零。 0x00：EDID 未读取 0x01：EDID 读取完成
305h~306h	rd and wr	htotal	2 bytes	视频分辨率参数 HTOTAL 配置 305h：htotal[15:8] 305h：htotal[7:0]
307h~308h	rd and wr	hsa	2 bytes	视频分辨率参数 HSA 配置 307h：hsa[15:8] 308h：hsa[7:0]
309h~30Ah	rd and wr	hfp	2 bytes	视频分辨率参数 HFP 配置 309h：hfp[15:8] 30Ah：hfp[7:0]
30Bh~30Ch	rd and wr	hbp	2 bytes	视频分辨率参数 HBP 配置

				30Bh：hbp[15:8] 30Ch：hbp[7:0]
30Dh~30Eh	rd and wr	hactive	2 bytes	视频分辨率参数 HACTIVE 配置 30Dh：hactive[15:8] 30Eh：hactive[7:0]
30Fh~310h	rd and wr	vtotal	2 bytes	视频分辨率参数 VTOTAL 配置 30Fh：vtotal[15:8] 310h：vtotal[7:0]
311h~312h	rd and wr	vsa	2 bytes	视频分辨率参数 VSA 配置 311h：vsa[15:8] 312h：vsa[7:0]
313h~314h	rd and wr	vfp	2 bytes	视频分辨率参数 VFP 配置 313h：vfp[15:8] 314h：vfp[7:0]
315h~316h	rd and wr	vbp	2 bytes	视频分辨率参数 VBP 配置 315h：vbp[15:8] 316h：vbp[7:0]
317h~318h	rd and wr	vactive	2 bytes	视频分辨率参数 VACTIVE 配置 317h：vactive[15:8] 318h：vactive[7:0]
319h	rd and wr	pixel_num	1 byte	像素模式选择 0x00：1 像素模式 0x01：2 像素模式 0x02：4 像素模式
31Ah	rd and wr	scramble_en	1 byte	加扰选择 0x00：禁用加扰 0x01：使能加扰
31Bh	rd and wr	tmds_clk_ratio	1 byte	数据与时钟速率比选择 0x00：10:1 0x01：40:1
31Ch	rd and wr	vsync_polarity	1 byte	VSYNC 极性选择 0x00：低有效 0x01：高有效
31Dh	rd and wr	hsync_polarity	1 byte	HSYNC 极性选择 0x00：低有效 0x01：高有效
31Eh	rd and wr	over_sample	1 byte	过采样模式选择 0x00：禁用过采样 0x01：使能 2 倍过采样 0x02：使能 4 倍过采样
31Fh	rd and wr	pixel_color_deep	1 byte	像素色深选择 0x00：24Bit 0x01：30Bit

320h	rd and wr	video_format	1 byte	视频格式选择 0x00：RGB 0x01：YUV444 0x02：YUV422 0x03：YUV420（保留，暂不支持）
321h	rd and wr	video_tpg_en	1 byte	内部测试画面使能，仅支持 RGB 视频格式 0x00：禁用内部测试画面 0x01：使能内部测试画面
322h	rd and wr	audio_channel_num	1 byte	音频通道数量选择 0x00：1 Audio Channel 0x01：2 Audio Channel 0x02：3 Audio Channel 0x03：4 Audio Channel 0x04：5 Audio Channel 0x05：6 Audio Channel 0x06：7 Audio Channel 0x07：8 Audio Channel
323h	rd only	hdmi_hdp	1 byte	HDMI HPD 输入 0x00：HPD 状态为 0 0x01：HPD 状态为 1

2.4.2 Video Interface

Video Interface 接口时序如图 2-6 所示， I_axis_s_video_user 信号是帧起始信号，为一个单时钟周期信号，在第一行第一个有效数据拉高。I_axis_s_video_valid 是数据有效信号，I_axis_s_video_last 是行结束信号，在一行数据的结尾拉高。O_axis_s_video_ready 是接收端准备信号，为高表示可以写入数据。

需要注意的是：O_axis_s_video_ready 只会在 I_axis_s_video_last 为高之后拉低，再次拉高表示能够正常写入数据，因此 Video Interface 不支持断续的数据输入，一行数据输入必须连续。

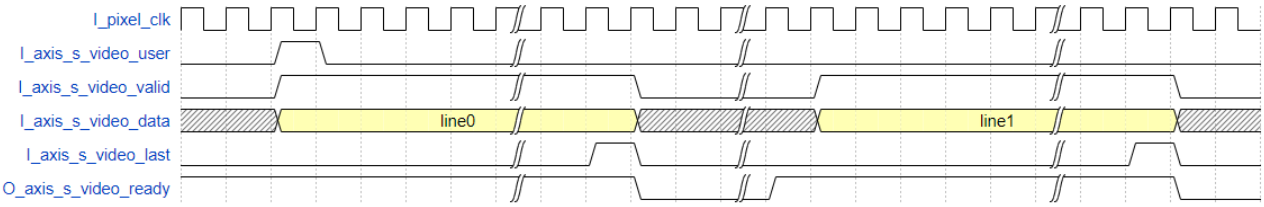

图 2-6 Video Interface 接口时序

在 HDMI 协议中，视频和辅助数据在发送的时候需要遵循严格的视频时序要求，为了保证发送视频时序的稳定，HDMI2.0 Transmitter Core 内置了 Video Source，用于生成不同分辨率的视频控制信号，然后视频数据和辅助数据先后映射到数据传输路径上。

由于视频数据的数据量大，无法把一帧数据缓存，因此把外部视频数据映射到 HDMI 数据路径上需要考虑相位问题，这个相位问题是 HDMI2.0 Transmitter Core 内部 Video Source 生成的视频数据和外部输入视频数据之间的偏移产生的。例如外部输入的视频数据超前或者滞后内部 Video Source 生成的视频数据，那么在进行数据映射的时候就会造成数据错位。

为了解决相位问题，HDMI2.0 Transmitter Core 内部设计了一个相位匹配机制，用于对内部和外部的视频时序进行相位校准，当内部和外部的视频时序对准并且锁定之后 O_video_locked 信号会拉高，表明视频输入数据被正确捕捉。

为了保证外部视频输入数据能够正确被映射到 HDMI 传输路径上，需要注意行长度 line_length、行间距 line_space 以及帧间距 frame_space 这三个参数，如图 2-7 所示。这三个参数和图像分辨率、像素模式以及像素深度有关。

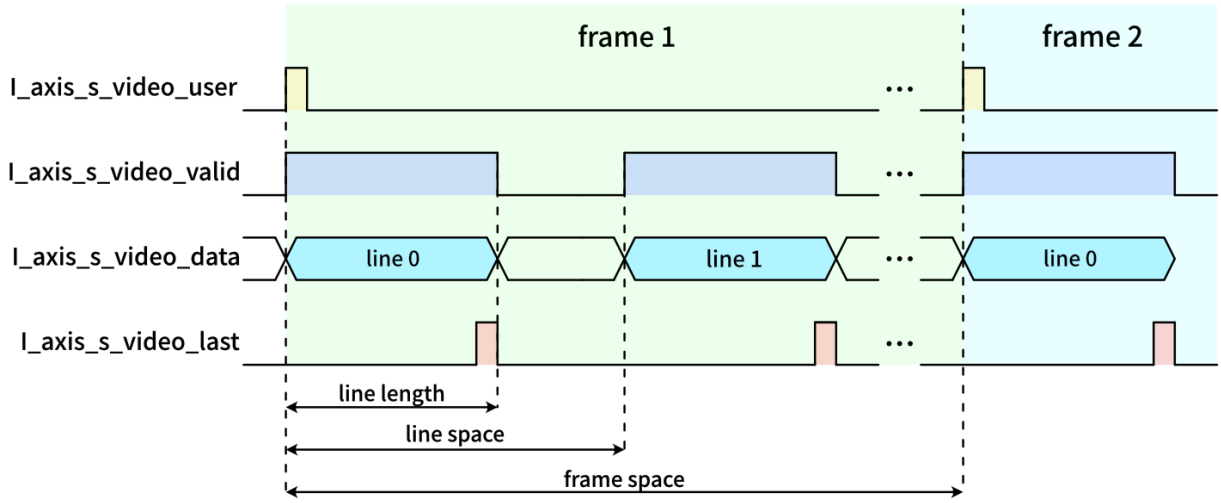

图 2-7 外部视频输入时序

图像一行总长度为 htotal，一行实际长度为 hactive，总的行数为 vtotal，实际行数为 vactive，表 2-4 列出了不同模式下 line_length、line_space 和 frame_space 的计算方法。

表 2-4 外部视频输入时序参数计算

像素模式	视频格式 / 参数	RGB 24bit YUV444 24bit	RGB 30bit YUV444 30bit	YUV422 24bit YUV422 30bit
1 像素模式	line_length	hactive	hactive	hactive
	line_space	htotal	htotal*1.25	htotal
	frame_space	vtotal	vtotal	vtotal
2 像素模式	line_length	hactive/2	hactive/2	hactive/2
	line_space	htotal/2	(htotal*1.25)/2	htotal/2
	frame_space	vtotal	vtotal	vtotal
4 像素模式	line_length	hactive/4	hactive/4	hactive/4
	line_space	htotal/4	(htotal*1.25)/4	htotal/4
	frame_space	vtotal	vtotal	vtotal

从表 2-4 可以看出，不同模式会影响图像水平方向的长度，多像素模式本质是一个时钟周期同时处理多个像素数据，因此 2 像素模式需要图像水平参数除以 2，4 像素模式需要图像水平参数除以 4。

RGB 和 YUV444 模式下，采用 30bit 色深的时候，相比于 24bit 色深多出了额外的数据，因此需要更多的时钟周期来容纳，图像水平方向的长度需要乘以 1.25 倍，由于 HDMI2.0 Transmitter Core 内部做了数据转换，因此外部视频数据输入的一行长度 line_length 无需扩长，只需要行间距 line_space 乘以 1.25 倍。

RGB 和 YUV444 在 30bit 色深的时候，为了达到和 24bit 色深一样的显示帧率，HDMI2.0 Transmitter 输出的线速率相比于 24bit 色深要乘以 1.25 倍。

以 4Kp60 显示分辨率为例，htotal 为 4400，hactive 为 3840，vtotal 为 2250，vactive 为 2160，

根据表 2-4 可以计算各个模式下外部视频输入的行长度、行间距以及帧间距。

表 2-5 4Kp60 分辨率外部输入参数

像素模式	视频格式 / 参数	RGB 24bit YUV444 24bit	RGB 30bit YUV444 30bit	YUV422 24bit YUV422 30bit
1 像素模式	line_length	3840	3840	3840
	line_space	4400	5500	4400
	frame_space	2250	2250	2250
2 像素模式	line_length	1920	1920	1920
	line_space	2200	2750	2200
	frame_space	2250	2250	2250
4 像素模式	line_length	960	960	960
	line_space	1100	1375	1100
	frame_space	2250	2250	2250

4 像素模式下视频数据的像素分布如图 2-8 所示。

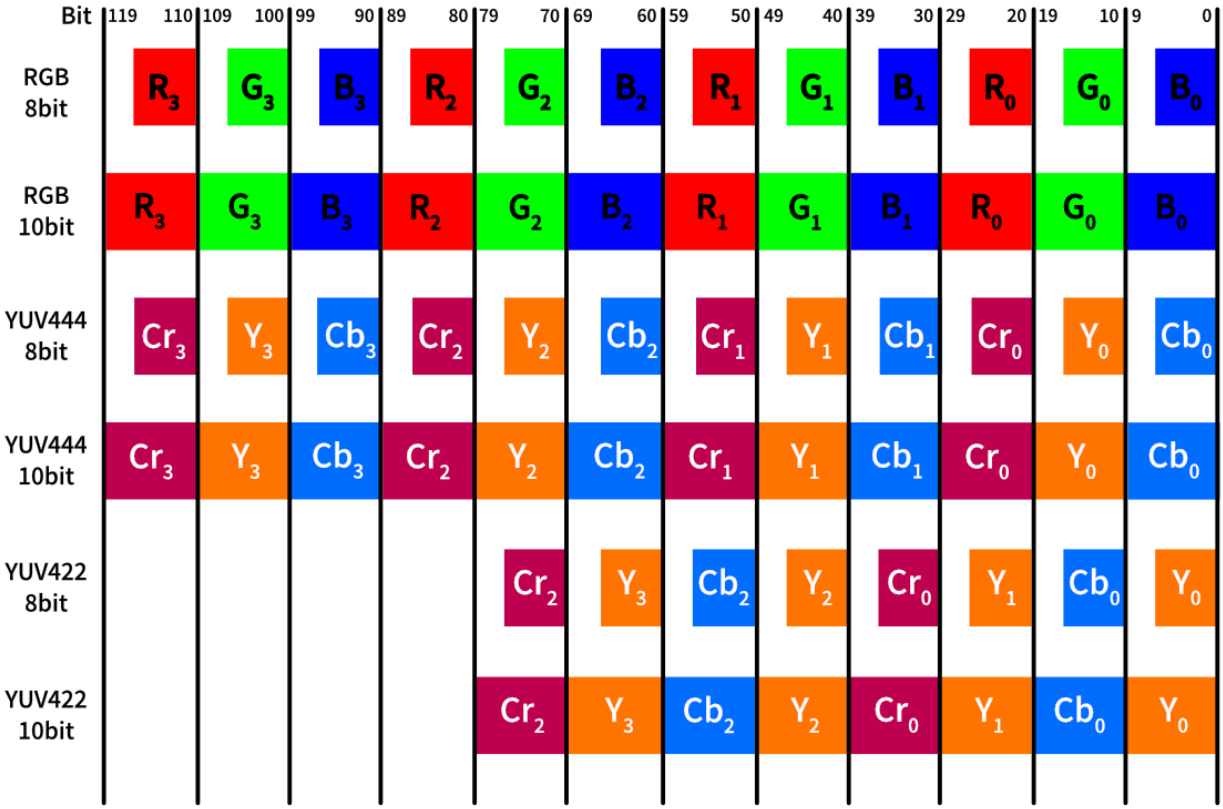

图 2-8 4 像素模式

2 像素模式下视频数据的像素分布如图 2-9 所示。

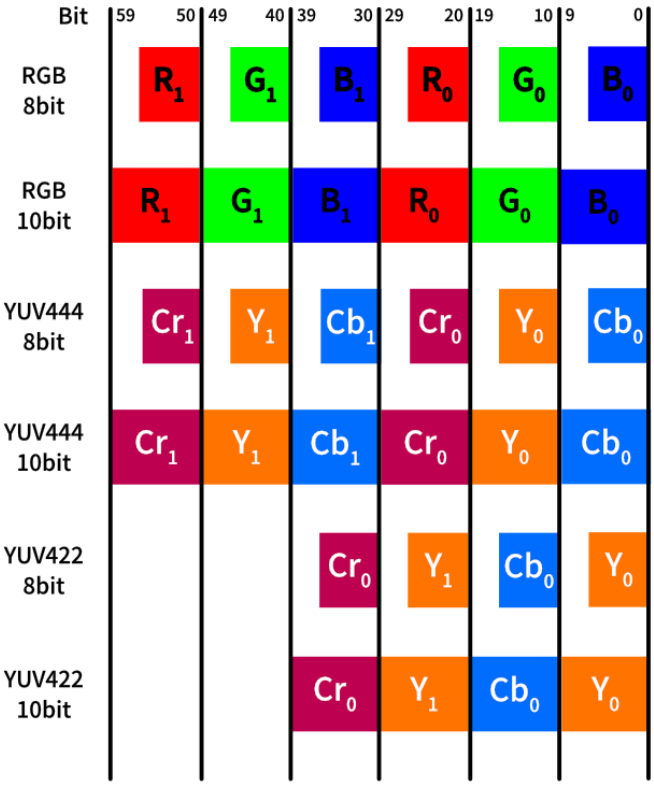

图 2-9 2 像素模式

1 像素模式下视频数据的像素分布如图 2-10 所示。

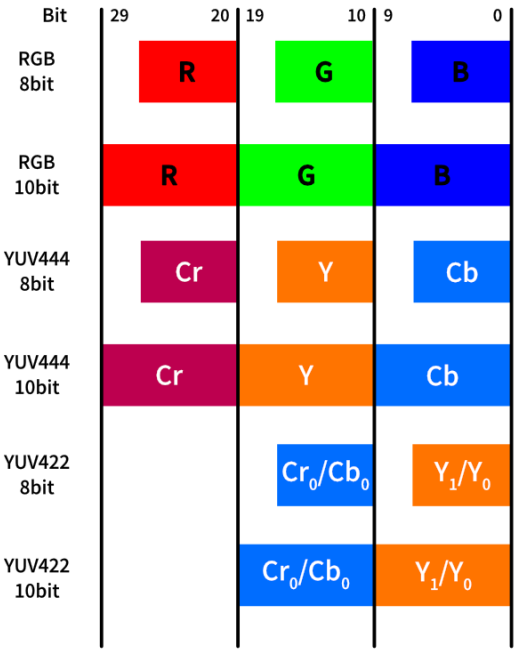

图 2-10 1 像素模式

YUV422 是 YUV444 舍弃了一些数据之后重新排布的，转换过程如图 2-11 所示。其中 Y 分量全采样，像素 00 和像素 01 共用像素 00 的 Cb 和 Cr 分量。

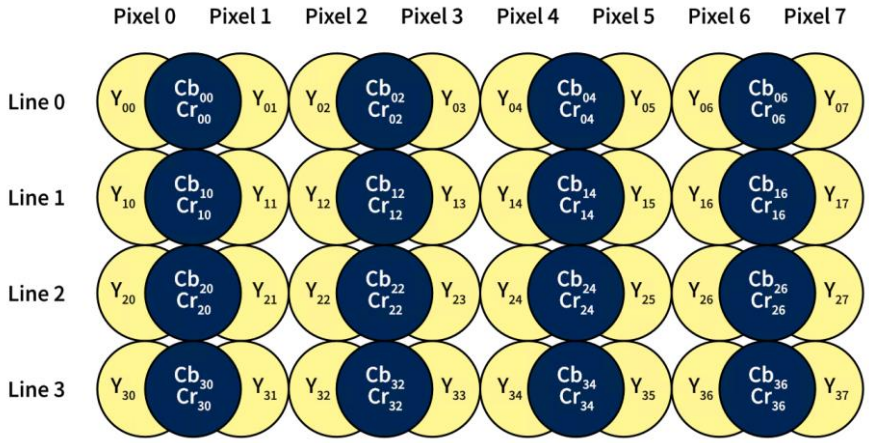

图 2-11 YUV444 转 YUV422

2.4.3 Audio Interface

在 HDMI2.0 Transmitter Core 中，音频采用 L-PCM（IEC60958,Packet Type 0x02）编码方式，最大支持 8 通道。在输入音频数据时，需要按照通道顺序输入，如果输入的音频通道是乱序的会引起异常。

Audio Interface 接口时序如图 2-12 所示，展示的是最大 8 通道的音频输入，实际使用时根据配置的通道数量决定输入的音频通道数量。单帧音频数据的传输中分为多个音频通道顺序发送，valid 是单周期信号，每次拉高一个时钟周期传输一个通道的音频数据，在一帧音频数据中 ready 信号在 valid 信号拉高之后会拉低一个周期，一帧音频数据发送结束时 ready 信号拉低的周期将大于一个时钟周期。

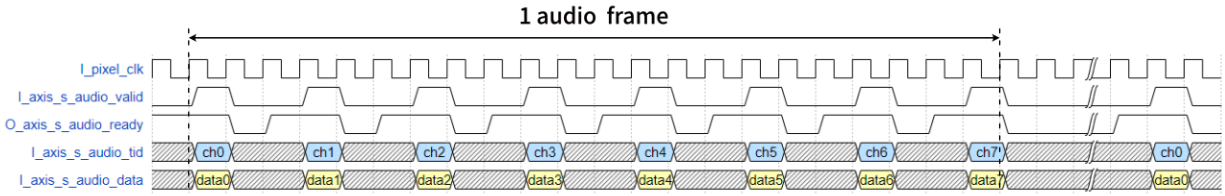

图 2-12 Audio Interface 接口时序

音频数据说明如表 2-6 所示。

表 2-6 音频数据说明

音频数据	说明
I_axis_s_audio_data[31:28]	Preamble Code 4'b0001:start of block 4'b0000:otherwise
I_axis_s_audio_data[27]	P(Parity Bit)
I_axis_s_audio_data[26]	C(Channel Status Bit)
I_axis_s_audio_data[25]	U(User Bit)
I_axis_s_audio_data[24]	V(Validity Bit)
I_axis_s_audio_data[23:0]	Audio Sample Data

P(Parity Bit)是奇偶校验位，当音频数据 Audio Sample Data 和通道状态位 C(Channel Status Bit)中的 bit1 的数量是奇数时需要设定奇偶校验位 P(Parity Bit)为 1，如果是偶数则 P(Parity Bit)为 0。

V(Validity Bit)是数据有效位，当为 0 是表示此次音频数据有效，为 1 时表示此次音频数据无效。

C(Channel Status Bit)是通道状态位，通道状态是一个 192bit 的数据，通过串行发送，每次音频数据通过该位发送一个 bit，在 Preamble Code 为 4'b0001 时作为串行发送起始。

U(User Bit)是用户定义数据，也是 192bit 数据，发送方式和通道状态同理。

通道状态 C(Channel Status Bit)和用户定义数据 U(User Bit)的具体含义参考 IEC 60958 协议标准。

通道信号 I_axis_s_audio_tid 是一个 3bit 数据，最大指示 8 通道，与音频通道的对应关系如表 2-7 所示。

表 2-7 TID 与音频通道的对应关系

I_axis_s_audio_tid[2:0]	Channel
3'b000	ch0
3'b001	ch1
3'b010	ch2
3'b011	ch3
3'b100	ch4
3'b101	ch5
3'b110	ch6
3'b111	ch7

2.4.4 Audio Clock Regeneration(ACR) Interface

Audio Clock Regenration(ACR)接口用于发送音频时钟再生辅助数据包，用于接收端从 TMDS 时钟中恢复音频时钟，更多有关 ACR 的信息请参见 HDMI1.4b 规范中的第 7 章。

接口时序如图 2-13 所示，I_acr_valid 为高时，HDMI2.0 Transmitter Core 从 ACR 接口读取 CTS 和 N 的值，然后发送音频时钟再生辅助数据包。

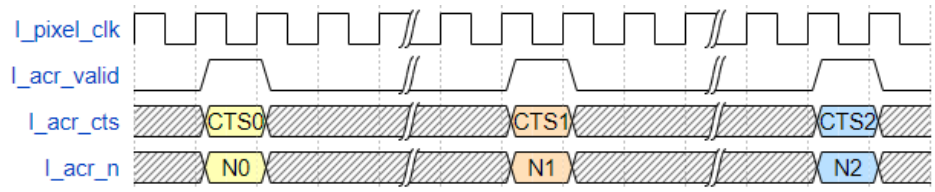

图 2-13 Audio Clock Regeneration(ACR)接口时序

3 配置驱动

HDMI2.0 Transmitter Core 内部的所有控制寄存器，以及 DDC 通道中 EDID、SCDC 的读写和 HPD 状态的读取都是通过 CPU 总线读写内部寄存器完成，寄存器列表见表 2-3。为此提供配套的软件驱动代码帮助使用者快速使用 HDMI2.0 Transmitter Core。

驱动代码包括 C 代码 anl_hdmi_tx.c 和头文件 anl_hdmi_tx.h，软件驱动代码在 Future Dynasty（FD）软件中运行，如图 3-1 所示，通过 FD 软件可以进行在线调试或者生成 mif 文件固化进 CPU 中。

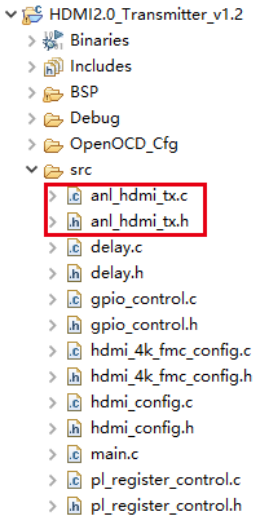

图 3-1 HDMI2.0 Transmitter Core 驱动代码

3.1 类型说明

在 anl_hdmi_tx.h 头文件中定义了驱动所有的数据类型，包括结构体和枚举类型。结构体用于映射底层寄存器和参数配置，枚举类型则根据 HDMI2.0 协议将寄存器的值转换为一组具有离散值的常量，便于程序的可读性和维护性。

定义的结构体如下所示：

```
1.  hdmi_tx_reg_type
2.  hdmi_tx_parameter_type
3.  hdmi_tx_general_control_packet_type
4.  hdmi_tx_vendor_specific_infoframe_packet_type
5.  hdmi_tx_avi_infoframe_packet_type
6.  hdmi_tx_spd_infoframe_packet_type
7.  hdmi_tx_audio_infoframe_packet_type
8.  hdmi_tx_scdc_setting_type
```

hdmi_tx_reg_type 结构体直接映射底层寄存器，在定义时设定结构体基地址为 CPU 总线起始地址。

hdmi_tx_parameter_type 结构体是 HDMI2.0 Transmitter Core 参数配置接口。

hdmi_tx_general_control_packet_type 结构体是 General Control 数据包的配置接口。

hdmi_tx_vendor_specific_infoframe_packet_type 结构体是 Vendor-Specific InfoFrame 数据包的配置接口。

hdmi_tx_avi_infoframe_packet_type 结构体是 AVI InfoFrame 数据包的配置接口。

hdmi_tx_spd_infoframe_packet_type 结构体是 Source Product Descriptor InfoFrame 数据包的配置接口。

hdmi_tx_audio_infoframe_packet_type 结构体是 Audio InfoFrame 数据包的配置接口。

hdmi_tx_scdc_setting_type 结构体是 SCDC 通道的配置接口。

3.2 驱动函数说明

HDMI2.0 Transmitter Core 提供的驱动函数如下所示：

```
1.  int anl_hdmi_tx_hpd_get(hdmi_tx_reg_type* hdmi_tx_reg);
2.  int anl_hdmi_tx_controller_parameter_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_parameter_type*
    hdmi_tx_parameter);
3.  int anl_hdmi_tx_general_control_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_general_contr
    ol_packet_type* general_control_packet);
4.  int anl_hdmi_tx_vendor_specific_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_ven
    dor_specific_infoframe_packet_type* vendor_specific_infoframe_packet);
5.  int anl_hdmi_tx_avi_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_avi_infoframe_p
    acket_type* avi_infoframe_packet);
6.  int anl_hdmi_tx_spd_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_spd_infoframe_p
    acket_type* spd_infoframe_packet);
7.  int anl_hdmi_tx_audio_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg, hdmi_tx_audio_infofr
    ame_packet_type* audio_infoframe_packet);
8.  int anl_hdmi_tx_scdc_setting_updata(hdmi_tx_reg_type* hdmi_tx_reg, hdmi_tx_scdc_setting_type* hdmi
    _tx_scdc_setting);
9.  int anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg_type* hdmi_tx_reg, enum hdmi_tx_aux_packet hdmi
    _aux_packet_type);
10. int anl_hdmi_tx_controller_reset_set(hdmi_tx_reg_type* hdmi_tx_reg);
11. int anl_hdmi_tx_controller_reset_release(hdmi_tx_reg_type* hdmi_tx_reg);
12. int anl_hdmi_tx_aux_packet_enable(hdmi_tx_reg_type* hdmi_tx_reg);
13. int anl_hdmi_tx_aux_packet_disable(hdmi_tx_reg_type* hdmi_tx_reg);
14. int anl_hdmi_tx_edid_read(hdmi_tx_reg_type* hdmi_tx_reg, uint8_t *edid_buffer);
15. int anl_hdmi_tx_scdc_read(hdmi_tx_reg_type* hdmi_tx_reg, uint8_t *scdc_buffer);
16. int anl_hdmi_tx_edid_read_printf(hdmi_tx_reg_type* hdmi_tx_reg);
17. int anl_hdmi_tx_scdc_read_printf(hdmi_tx_reg_type* hdmi_tx_reg);
```

各个函数的说明如下：

```
/*
* 函数功能：读取 HDMI HPD 信号状态
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0：HPD=0
*       1：HPD=1
*/
int anl_hdmi_tx_hpd_get(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：将 HDMI2.0 Transmitter Core 的配置参数写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       hdmi_tx_parameter ：hdmi_tx_parameter_type 结构体类型，填入 HDMI2.0 Transmitter Core 的
*                           配置参数
* 返回：0
*/
int anl_hdmi_tx_controller_parameter_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_parameter_type*
hdmi_tx_parameter);
```

```
/*
* 函数功能：将 General Control 数据包写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       general_control_packet ：hdmi_tx_general_control_packet_type 结构体类型，填入
*                                 General Control 数据包
* 返回：0
*/
int anl_hdmi_tx_general_control_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_general_contr
ol_packet_type* general_control_packet);
```

```
/*
* 函数功能：将 Vendor Specific InfoFrame 数据包写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       vendor_specific_infoframe_packet :hdmi_tx_vendor_specific_infoframe_packet_type 结构体类
*                                         型，填入 Vendor Specific InfoFrame 数据包
* 返回：0
*/
int anl_hdmi_tx_vendor_specific_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_ven
dor_specific_infoframe_packet_type* vendor_specific_infoframe_packet);
```

```
/*
* 函数功能：将 AVI InfoFrame 数据包写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       avi_infoframe_packet ：hdmi_tx_avi_infoframe_packet_type 结构体类型，填入
*                              AVI InfoFrame 数据包内容
* 返回：0
*/
int anl_hdmi_tx_avi_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_avi_infoframe_packet_type* avi_infoframe_packet);
```

```
/*
* 函数功能：将 Source Product Descriptor InfoFrame 数据包写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       spd_infoframe_packet ：hdmi_tx_spd_infoframe_packet_type 结构体类型，填入
*                              Audio InfoFrame 数据包内容
* 返回：0
*/
int anl_hdmi_tx_spd_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg,hdmi_tx_spd_infoframe_packet_type* spd_infoframe_packet);
```

```
/*
* 函数功能：将 Audio InfoFrame 数据包写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       audio_infoframe_packet ：hdmi_tx_audio_infoframe_packet_type 结构体类型，填入
*                                Audio InfoFrame 数据包内容
* 返回：0
*/
int anl_hdmi_tx_audio_infoframe_packet_updata(hdmi_tx_reg_type* hdmi_tx_reg, hdmi_tx_audio_infoframe_packet_type* audio_infoframe_packet);
```

```
/*
* 函数功能：将 SCDC 设置写入到寄存器中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       hdmi_tx_scdc_setting ：hdmi_tx_scdc_setting_type 结构体类型，填入设定好的 SCDC 数据
* 返回：0
*/
int anl_hdmi_tx_scdc_setting_updata(hdmi_tx_reg_type* hdmi_tx_reg, hdmi_tx_scdc_setting_type* hdmi_tx_scdc_setting);
```

```
/*
* 函数功能：从寄存器中将指定的辅助数据包读取出来并通过 printf 函数打印到中端
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       hdmi_aux_packet_type：hdmi_tx_aux_packet 枚举体类型，设定辅助数据包类型
* 返回：0
*/
int anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg_type* hdmi_tx_reg, enum hdmi_tx_aux_packet hdmi_aux_packet_type);
```

```
/*
* 函数功能：对 HDMI2.0 Transmitter Core 进行复位操作，该操作不会释放
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_controller_reset_set(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：对 HDMI2.0 Transmitter Core 进行释放复位操作
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_controller_reset_release(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：打开辅助数据包发送使能
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_aux_packet_enable(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：关闭辅助数据包发送使能
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_aux_packet_disable(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：从 HDMI Sink 端读取 EDID 数据放到 edid_buffer 中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       edid_buffer：深度为 256Byte 的数组首地址，用于存放读取的 EDID 数据
* 返回：0
*/
int anl_hdmi_tx_edid_read(hdmi_tx_reg_type* hdmi_tx_reg, uint8_t *edid_buffer);
```

```
/*
* 函数功能：从 HDMI Sink 端读取 SCDC 数据放到 edid_buffer 中
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
*       scdc_buffer：深度为 256Byte 的数组首地址，用于存放读取的 SCDC 数据
* 返回：0
*/
int anl_hdmi_tx_scdc_read(hdmi_tx_reg_type* hdmi_tx_reg, uint8_t *scdc_buffer);
```

```
/*
* 函数功能：从 HDMI Sink 端读取 EDID 数据通过 printf 函数打印到终端
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_edid_read_printf(hdmi_tx_reg_type* hdmi_tx_reg);
```

```
/*
* 函数功能：从 HDMI Sink 端读取 SCDC 数据通过 printf 函数打印到终端
* 输入：hdmi_tx_reg：hdmi_tx_reg_type 结构体，基地址为 CPU 总线起始地址
* 返回：0
*/
int anl_hdmi_tx_scdc_read_printf(hdmi_tx_reg_type* hdmi_tx_reg);
```

3.3 参数配置说明

HDMI2.0 Transmitter Core 的参数配置分为两步：

首先定义 hdmi_tx_parameter_type 结构体，配置结构体中每项的参数，然后通过 anl_hdmi_tx_controller_parameter_updata 函数将配置参数写入到寄存器中。

```
typedef struct{
    uint16_t htotal;
    uint16_t hsa;
    uint16_t hfp;
    uint16_t hbp;
    uint16_t hactive;
    uint16_t vtotal;
    uint16_t vsa;
    uint16_t vfp;
    uint16_t vbp;
    uint16_t vactive;
    enum hdmi_tx_pixel_num          pixel_num;
    enum hdmi_tx_scramble_en        scramble_en;
    enum hdmi_tx_tmds_clk_ratio     tmds_clk_ratio;
    enum hdmi_tx_vsync_polarity     vsync_polarity;
    enum hdmi_tx_hsync_polarity     hsync_polarity;
    enum hdmi_tx_over_sample        over_sample;
    enum hdmi_tx_pixel_color_depth  pixel_color_depth;
    enum hdmi_tx_video_format       video_format;
    enum hdmi_tx_video_tpg_en       video_tpg_en;
    enum hdmi_tx_audio_channel_num  audio_channel_num;
}hdmi_tx_parameter_type;
```

3.3.1 显示分辨率配置

htotal、hsa、hfb、hbp、hactive、vtotal、vsa、vfp、vbp、vactive 用于配置显示分辨率，视频时序如图 3-2 所示。

需要注意的是：配置的水平分辨率参数要能够被对应的像素模式整除，比如 4 像素模式下所有的水平分辨率参数 htotal、hsa、hfp、hbp 要能够被 4 整除。

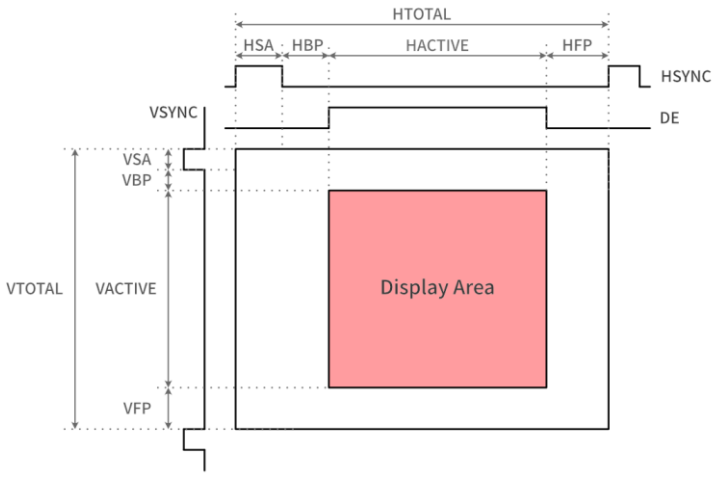

图 3-2 视频时序

3.3.2 多像素模式

pixel_num 用来设定每个像素时钟下处理的像素个数，可选项如下所示：

```
1.  enum hdmi_tx_pixel_num {
2.      HDMI_TX_1_PIXEL = 0,  /*1 像素模式*/
3.      HDMI_TX_2_PIXEL = 1,  /*2 像素模式*/
4.      HDMI_TX_4_PIXEL = 2,  /*4 像素模式*/
5.  };
```

当设定每个时钟处理 2 两个像素时，意味着视频输入接口一个时钟周期下需要输入两个像素，HDMI2.0 Transmitter Core 内部会同时对这两个像素数据进行打包编码，4 像素模式同理可推。在 1 像素模式下 HDMI2.0 Transmitter Core 内部 TMDS 编码器输出的数据位宽为 10Bit，2 像素模式下输出的数据位宽为 20Bit，4 像素模式下输出的数据位宽为 40Bit，因此不同的像素模式需要和 SerDes 的外部数据位宽相匹配。

多像素模式是为了解决高分辨率下像素时钟频率过高引发的时序问题，以 4Kp60 分辨率为例，输出线速率为 5.94Gbps，像素时钟频率为 594MHz，FPGA 内部是无法如此高的频率下处理图像数据的。因此通过设置 SerDes 数据位宽和像素模式，可以有效降低像素时钟频率。

3.3.3 加扰模式

scramble_en 用于设定是否启动加扰模式，可选项如下所示：

```
1.  enum hdmi_tx_scramble_en {
2.      HDMI_TX_SCRAMBLE_DISABLE = 0,  /*禁用加扰*/
3.      HDMI_TX_SCRAMBLE_ENABLE = 1,   /*使能加扰*/
4.  };
```

根据 HDMI2.0 协议，当源端输出线速率大于 3.4Gbps 时应该启动加扰模式。源端输出线速率小于等于 3.4Gbps 时，一般选择关闭加扰模式，如果接收端支持低线速率解扰也可开启加扰。

3.3.4 TMDS 时钟比例

tmds_clk_ratio 用来设置数据和时钟速率的比例关系，可选项如下：

```
1.  enum hdmi_tx_tmds_clk_ratio {
2.      HDMI_TX_CLOCK_RATIO_1_10 = 0,  /*数据和时钟速率比例为 10:1*/
3.      HDMI_TX_CLOCK_RATIO_1_40 = 1,  /*数据和时钟速率比例为 40:1*/
4.  };
```

根据 HDMI2.0 协议，当源端输出线速率大于 3.4Gbps 时，需要设置数据和时钟速率的比例为 40:1，如果源端输出线速率小于等于 3.4Gbps，设置数据和时钟速率的比例为 10:1。

3.3.5 行场信号极性

vsync_polarity 和 hsync_polarity 用于设定行场信号 VSYNC 和 HSYNC 的极性，可选项如下所示：

```
enum hdmi_tx_vsync_polarity {
    HDMI_TX_VSYNC_LOW_VALID = 0,   /*VSYNC 低有效*/
    HDMI_TX_VSYNC_HIGH_VALID = 1,  /*VSYNC 高有效*/
};
```

```
enum hdmi_tx_hsync_polarity {
    HDMI_TX_HSYNC_LOW_VALID = 0,   /*HSYNC 低有效*/
    HDMI_TX_HSYNC_HIGH_VALID = 1,  /*HSYNC 高有效*/
};
```

3.3.6 过采样

over_sample 用于设定过采样类型，可选项如下所示：

```
enum hdmi_tx_over_sample {
    HDMI_TX_OVER_SAMPLE_DISABLE = 0,  /*禁用过采样*/
    HDMI_TX_2_TIMES_OVER_SAMPLE = 1,  /*2 倍过采样使能*/
    HDMI_TX_4_TIMES_OVER_SAMPLE = 2,  /*4 倍过采样使能*/
};
```

为了支持低线速率的视频分辨率，HDMI2.0 Transmitter Core 设置了过采样功能，当开启过采样时，控制器内部会对 TMDS 编码器输出的数据进行位复制，使得输出的线速率降低。

表 3-1 列出了不同像素模式以及过采样是否开启的情况下，所对应的 SerDes 外部数据位宽以及像素时钟频率的计算，其中 linerate 是指 HDMI 输出的线速率。

表 3-1 过采样模式下不同像素模式和 SerDes 外部位宽对应关系

过采样	像素模式	TMDS 输出位宽	SerDes 外部数据位宽	像素时钟频率
禁用	1 PIXEL	10Bit	10Bit	linerate/10
	2 PIXEL	20Bit	20Bit	linerate/20
	4 PIXEL	40Bit	40Bit	linerate/40
2 倍过采样	1 PIXEL	10Bit	20Bit	linerate/20
	2 PIXEL	20Bit	40Bit	linerate/40
4 倍过采样	1 PIXEL	10Bit	40Bit	linerate/40

以 720p60 分辨率为例，线速率为 0.7425Gbps，SerDes 无法输出如此低的线速率，为此可以把 SerDes 线速率设为 1.485Gbps，外部位宽设为 20Bit，像素模式设为 1 像素，然后开启 2 倍过采样，输出的线

速率就为 1.485Gbps/2=0.7425Gbps。

需要注意的是：过采样功能会对 TMDS 编码器输出的数据位宽进行拓宽，目前编码器输出与 SerDes 对接的位宽最大为 40Bit，因此在 1 像素模式下支持 2 倍过采样和 4 倍过采样，2 像素模式下只支持 2 倍过采样。

3.3.7 像素深度

pixel_color_depth 用于设定像素深度，可选项如下所示：

```
1.  enum hdmi_tx_pixel_color_depth {
2.      HDMI_TX_COLOR_DEPTH_24BIT = 0,  /*24Bit 色深*/
3.      HDMI_TX_COLOR_DEPTH_30BIT = 1,  /*30Bit 色深*/
4.  };
```

以 RGB 为例，色深设为 24Bit 时，RGB 三个通道 R、G、B 位宽为 8Bit，当色深设为 30Bit 时，RGB 三个通道 R、G、B 位宽为 10bit。**需要注意的是色深设为 30Bit 时需要 HDMI 接收端支持，否则会引起显示异常。**

3.3.8 视频格式

video_format 用于设定视频格式，可选项如下所示：

```
1.  enum hdmi_tx_video_format {
2.      HDMI_TX_VIDEO_FORMAT_RGB   = 0,     /*RGB 格式*/
3.      HDMI_TX_VIDEO_FORMAT_YUV444 = 1,  /*YUV444 格式*/
4.      HDMI_TX_VIDEO_FORMAT_YUV422 = 2,  /*YUV422 格式*/
5.      HDMI_TX_VIDEO_FORMAT_YUV420 = 3,  /*保留，YUV420 暂未支持*/
6.  };
```

3.3.9 内部视频测试画面

video_tpg_en 用于设定是否开启内部测试画面，可选项如下所示：

```
1.  enum hdmi_tx_video_tpg_en {
2.      HDMI_TX_VIDEO_TPG_DISABLE = 0,  /*内部测试画面关闭*/
3.      HDMI_TX_VIDEO_TPG_ENABLE = 1,   /*内部测试画面开启*/
4.  };
```

当内部测试画面开启时，HDMI2.0 Transmitter Core 内置的测试画面会按照图 3-3 所示的显示顺序播放测试画面，此时 Video Interface 输入的视频数据无效，关闭内部测试画面时 Video Interface 输入视频数据有效。

需要注意的是内部测试画面只支持 RGB 模式，其他视频格式开启内部测试画面时会引起显示异常。

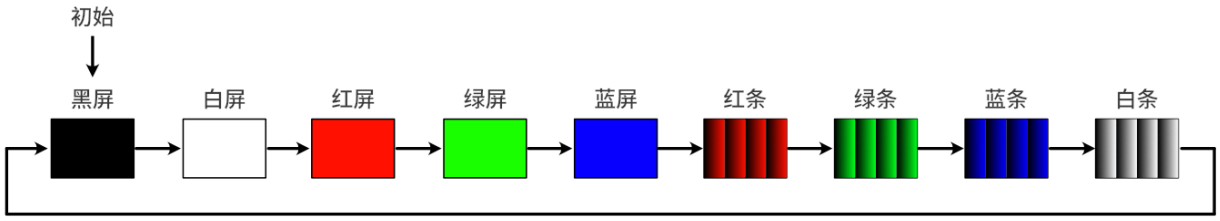

图 3-3 测试视频画面

3.3.10 音频数据通道

audio_channel_num 用于设定 Audio Interface 输入的音频数据通道数量，可选项如下所示：

```
enum hdmi_tx_audio_channel_num {
        HDMI_TX_AUDIO_1_CHANNEL = 0,  /*音频通道数为 1*/
        HDMI_TX_AUDIO_2_CHANNEL = 1,  /*音频通道数为 2*/
        HDMI_TX_AUDIO_3_CHANNEL = 2,  /*音频通道数为 3*/
        HDMI_TX_AUDIO_4_CHANNEL = 3,  /*音频通道数为 4*/
        HDMI_TX_AUDIO_5_CHANNEL = 4,  /*音频通道数为 5*/
        HDMI_TX_AUDIO_6_CHANNEL = 5,  /*音频通道数为 6*/
        HDMI_TX_AUDIO_7_CHANNEL = 6,  /*音频通道数为 7*/
        HDMI_TX_AUDIO_8_CHANNEL = 7,  /*音频通道数为 8*/
};
```

需要注意的是：当软件中设定了音频输入通道数量之后，Audio Interface 输入的音频通道数量需要与之匹配，否则会引起音频数据异常。

4 SerDes 配置

HDMI2.0 Transmitter 的物理层采用 SerDes 实现，以 PH1A 系列器件的 SerDes 为例，HDMI2.0 Transmitter Core 与 SerDes 的连接关系如图 4-1 所示。

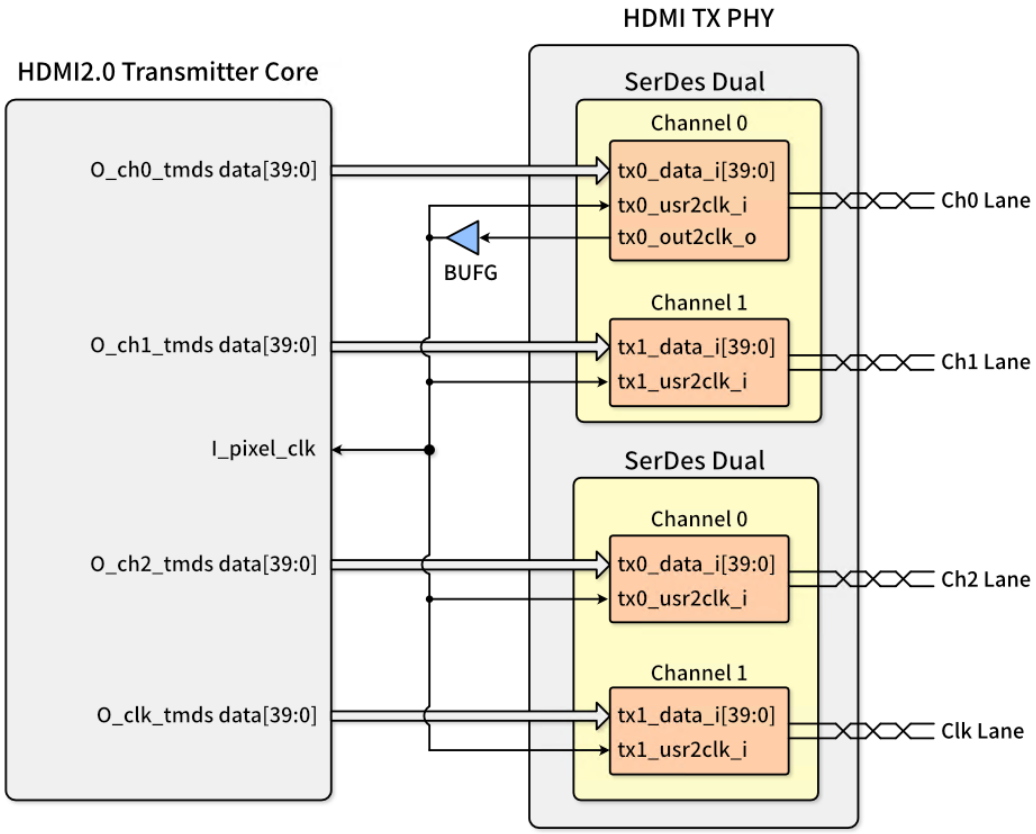

图 4-1 HDMI2.0 Transmitter Core 与 SerDes 的连接方式

PH1A 系列 FPGA 中的 SerDes 以 SerDes Dual 的形式分布在 FPGA 中，每个 SerDes Dual 中包含两个 Channel，即 Channel0 和 Channel1，每个 Channel 都有发送和接收，在 HDMI Transmitter 中只使用发送。

对于 HDMI Transmitter 来说，需要 4 个 Channel 来发送 HDMI 的数据和时钟，为此需要两个 SerDes Dual 来实现。

时钟方面采用任意一个 Channel 的 out2clk 作为 HDMI2.0 Transmitter Core 的像素时钟以及每个 SerDes Channel 的 usr2clk。

PH1A 器件 SerDes Protocol and Dual 的 IP 配置界面如图 4-2 所示。

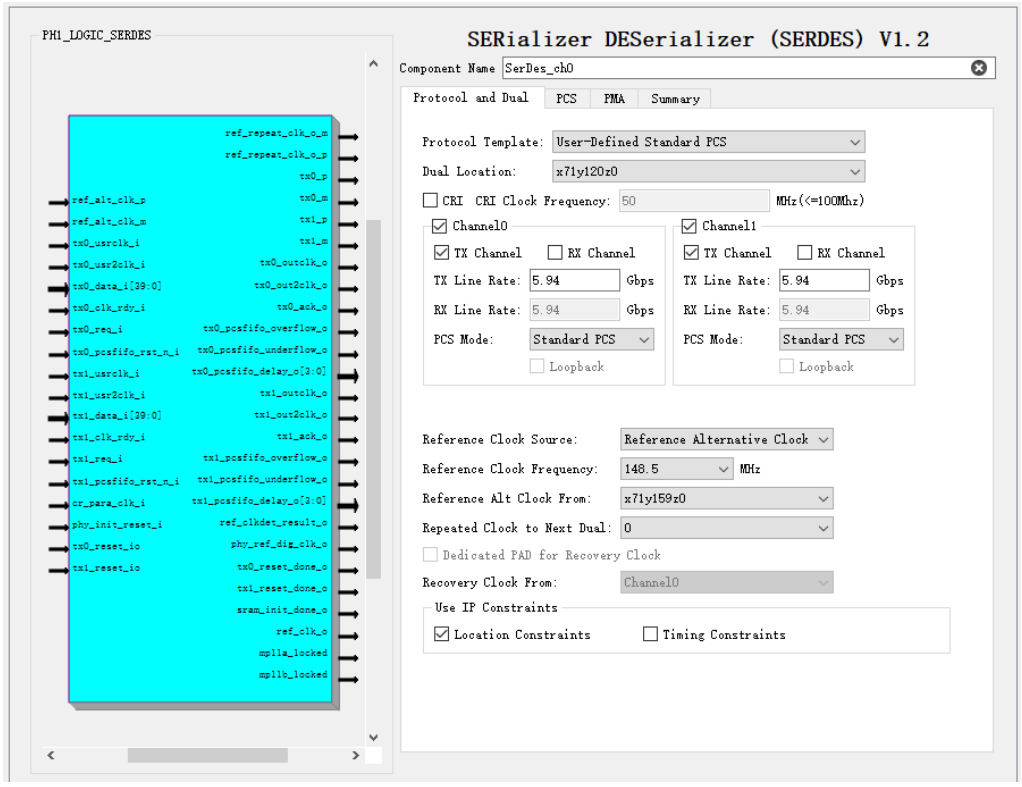

图 4-2 SerDes Protocol and Dual 配置界面

PCS 的 IP 配置界面如图 4-3 所示。

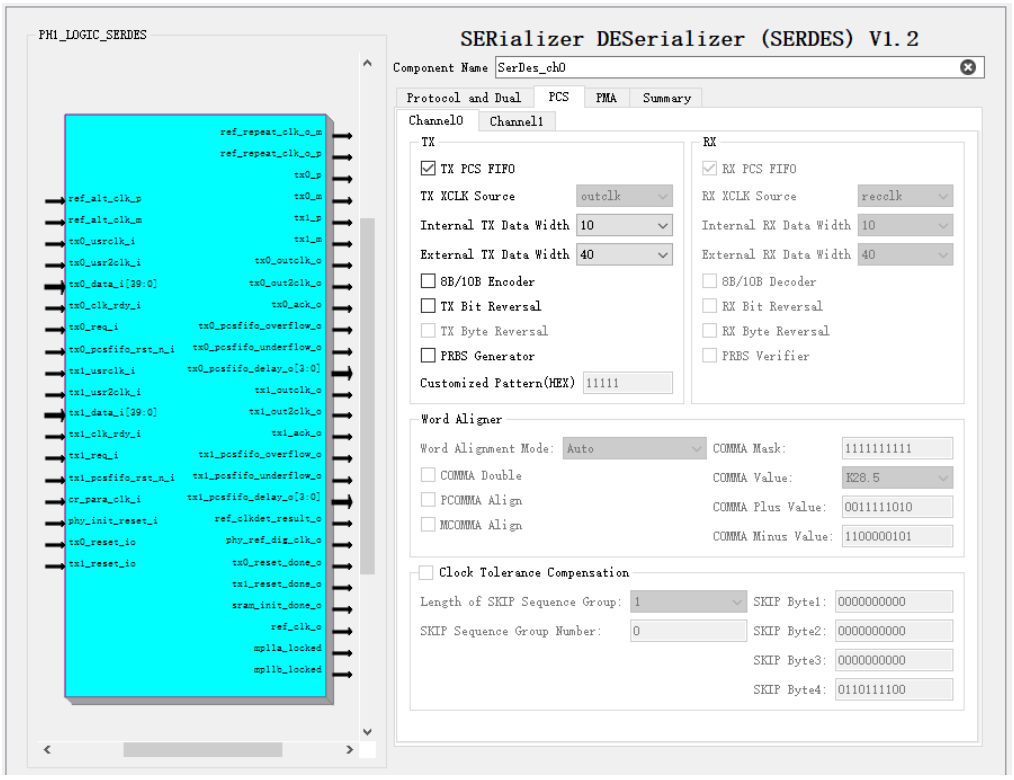

图 4-3 SerDes PCS 配置界面

PMA 的 IP 配置界面如图 4-4 所示。

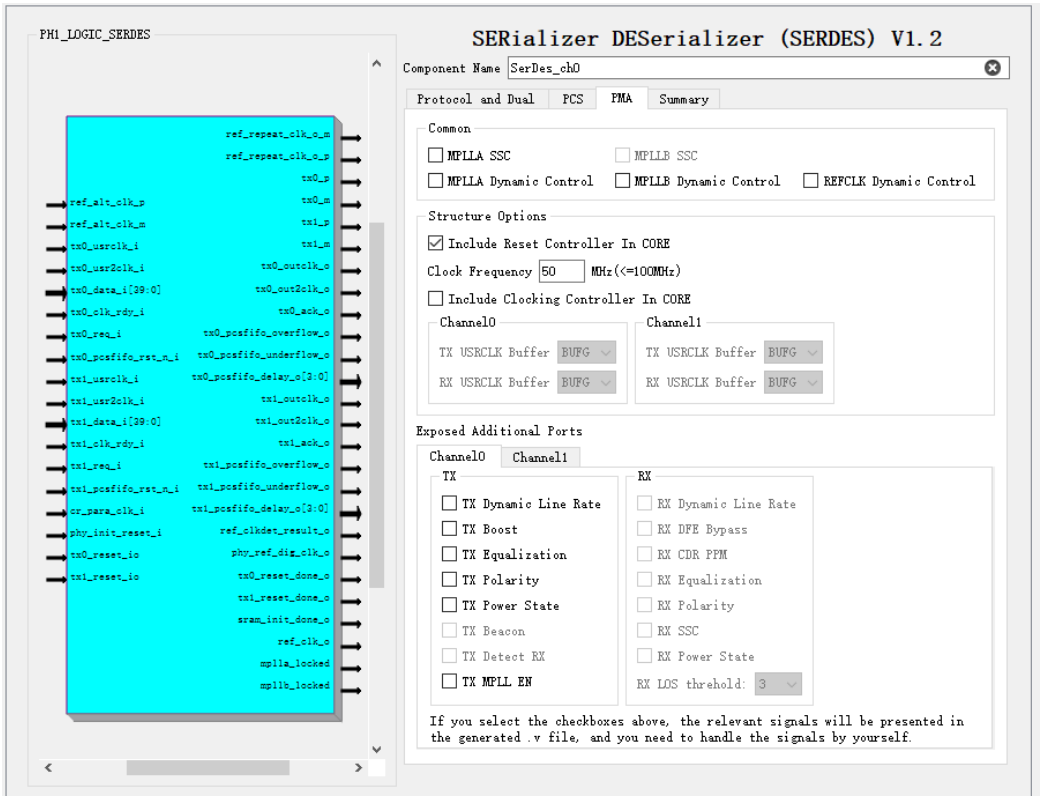

图 4-4 SerDes PMA 配置界面

5 HDMI2.0 Transmitter 配置流程示例

5.1 配置参数计算

HDMI Transmitter 在工作之前需要配置 SerDes 的线速率以及 HDMI2.0 Transmitter Core 的参数，下面以 4Kp60 显示分辨率为例详细说明参数的计算过程。

5.1.1 SerDes 线速率计算

4Kp60 分辨率在《ANSI/CTA-861-H》标准上可以查到图像的实际长宽为 4400x2250，如图 5-1 所示。因此可以计算出像素时钟频率如下，其中 frame_rate 为显示帧率。

$$f_{pixel\_clk} = HTOTAL \times VTOTAL \times frame\_rate = 4400 \times 2250 \times 60 = 594MHz$$

Field Rate[5]									(kHz)	(Hz)	(MHz)
	VIC	Hactive	Vactive	I / P	Htotal	Hblank[5]	Vtotal	Vblank[5]	H Freq[5]	V Freq[4]	Pixel Freq[5]
60Hz[3]	1	640	480	Prog	800	160	525	45	31.469	59.940[3]	25.175
	2, 3	720	480	Prog	858	138	525	45	31.469	59.940[3]	27.000
	4, 69	1280	720	Prog	1650	370	750	30	45.000	60.000[3]	74.250
	5	1920	1080	Int	2200	280	1125	22.5[1]	33.750	60.000[3]	74.250
	6, 7	1440[2]	480	Int	1716[2]	276	525	22.5[1]	15.734	59.940[3]	27.000
	8, 9	1440[2]	240	Prog	1716[2]	276	262	22	15.734	60.054[3]	27.000
	8, 9	1440[2]	240	Prog	1716[2]	276	263	23	15.734	59.826[3]	27.000
	10, 11	2880[2]	480	Int	3432[2]	552	525	22.5[1]	15.734	59.940[3]	54.000
	12, 13	2880[2]	240	Prog	3432[2]	552	262	22	15.734	60.054[3]	54.000
	12, 13	2880[2]	240	Prog	3432[2]	552	263	23	15.734	59.826[3]	54.000
	14, 15	1440[2]	480	Prog	1716[2]	276	525	45	31.469	59.940[3]	54.000
	16, 76	1920	1080	Prog	2200	280	1125	45	67.500	60.000[3]	148.500
	35, 36	2880[2]	480	Prog	3432[2]	552	525	45	31.469	59.940[3]	108.000
	83	1680	720	Prog	2200	520	750	30	45.000	60.000[3]	99.000
	90	2560	1080	Prog	3000	440	1100	20	66.000	60.000[3]	198.000
	97, 107	3840	2160	Prog	4400	560	2250	90	135.000	60.000[3]	594.000
	102	4096	2160	Prog	4400	304	2250	90	135.000	60.000[3]	594.000
	126	5120	2160	Prog	5500	380	2250	90	135.000	60.000[3]	742.500
	199, 207	7680	4320	Prog	9000	1320	4400	80	264.000	60.000[3]	2376.000
	215	10240	4320	Prog	11000	760	4500	180	270.000	60.000[3]	2970.000

图 5-1 4Kp60 视频分辨率格式

可以算出 4Kp60 分辨率的像素时钟频率为 594MHz，HDMI 输出的线速率是像素时钟频率的 10 倍，因此输出线速率为：

$$linerate = f_{pixel\_clk} \times 10 = 5.94Gbps$$

因此 SerDes 输出的线速率为 5.94Gbps，根据这个线速率来配置 SerDes 的输出线速率。SerDes 采用内部位宽 10Bit，外部位宽 40Bit，这样像素时钟频率为 594MHz/4=148.5MHz。

5.1.2 HDMI2.0 Transmitter Core 配置参数计算

4Kp60 分辨率在《ANSI/CTA-861-H》标准上可以查到图像的具体时序信息，如图 5-2 所示，根据这些信息配置得到分辨率参数如下：

htotal=4400；has=88；hfb=176；hbp=296；hactive=3840；vtotal=2250；vsa=10；vfp=8；vbp=72；vactive=2160。

Field Rate[28]	VIC	Fig	Hfront	Hsync	Hback	Hpol[18]	Vfront	Vsync	Vback	Vpol[18]	Ln	Reference Standard	Notes
60Hz	1	1	16	96	48	N	10	2	33	N	1	VESA DMT [124]	3, 4
	2, 3	1	16	62	60	N	9	6	30	N	7	CTA-770.2 [30]	2
	4, 69	2	110	40	220	P	5	5	20	P	1	CTA-770.3 [31]	1, 2
	5	4	88	44	148	P	2	5	15	P	1	CTA-770.3 [31]	1, 2
	6, 7	3	38	124	114	N	4	3	15	N	4	CTA-770.2 [30]	2, 15
	8, 9	1	38	124	114	N	4[20]	3	15	N	4	CTA-770.2 [30][17]	7, 14, 15, 19
	8, 9	1	38	124	114	N	5[21]	3	15	N	4	CTA-770.2 [30][17]	7, 14, 15, 19
	10, 11	3	76	248	228	N	4	3	15	N	4	CTA-770.2 [30][17]	8, 13
	12, 13	1	76	248	228	N	4[20]	3	15	N	4	CTA-770.2 [30][17]	7, 8, 13, 19
	12, 13	1	76	248	228	N	5[21]	3	15	N	4	CTA-770.2 [30][17]	7, 8, 13, 19
	14, 15	1	32	124	120	N	9	6	30	N	7	CTA-770.2 [30]	9, 10, 13, 14
	16, 76	2	88	44	148	P	4	5	36	P	1	SMPTE ST 274 [2]	14
	35, 36	1	64	248	240	N	9	6	30	N	7	CTA-770.2 [30]	9, 11
	83	2	260	40	220	P	5	5	20	P	1	CTA-770.3[31]	1, 2, 26
	90	2	248	44	148	P	4	5	11	P	1	SMPTE ST 274 [2]	14, 27
	97, 107	2	176	88	296	P	8	10	72	P	1	SMPTE ST 274 [2]	1, 2, 29
	102	2	88	88	128	P	8	10	72	P	1	SMPTE ST 274 [2]	1, 2, 30
	126	2	164	88	128	P	8	10	72	P	1	SMPTE ST 274 [2]	31
	199, 207	2	552	176	592	P	16	20	44	P	1	SMPTE ST 274 [2]	
	215	2	288	176	296	P	16	20	144	P	1	SMPTE ST 274 [2]	32

图 5-2 4Kp60 分辨率时序

HDMI2.0 Transmitter Core 4Kp60 分辨率的参数配置如下所示：

```
1.  hdmi_tx_parameter.htotal = 4400;
2.  hdmi_tx_parameter.hsa   = 88;
3.  hdmi_tx_parameter.hfp   = 176;
4.  hdmi_tx_parameter.hbp   = 296;
5.  hdmi_tx_parameter.hactive = 3840;
6.  hdmi_tx_parameter.vtotal = 2250;
7.  hdmi_tx_parameter.vsa   = 10;
8.  hdmi_tx_parameter.vfp   = 8;
9.  hdmi_tx_parameter.vbp   = 72;
10. hdmi_tx_parameter.vactive = 2160;
11. hdmi_tx_parameter.pixel_num = HDMI_TX_4_PIXEL;
12. hdmi_tx_parameter.scramble_en = HDMI_TX_SCRAMBLE_ENABLE;
13. hdmi_tx_parameter.tmds_clk_ratio = HDMI_TX_CLOCK_RATIO_1_40;
14. hdmi_tx_parameter.vsync_polarity = HDMI_TX_VSYNC_HIGH_VALID;
15. hdmi_tx_parameter.hsync_polarity = HDMI_TX_HSYNC_HIGH_VALID;
16. hdmi_tx_parameter.over_sample = HDMI_TX_OVER_SAMPLE_DISABLE;
17. hdmi_tx_parameter.pixel_color_depth = HDMI_TX_COLOR_DEPTH_24BIT;
```

```
hdmi_tx_parameter.video_format = HDMI_TX_VIDEO_FORMAT_RGB;
hdmi_tx_parameter.video_tpg_en = HDMI_TX_VIDEO_TPG_ENABLE;
hdmi_tx_parameter.audio_channel_num = HDMI_TX_AUDIO_2_CHANNEL;
```

5.2 配置代码

HDMI2.0 Transmitter Core 4Kp60 分辨率完整的配置代码如下，源代码见提供的软件工程中的 hdmi_config.c 文件。

```
/*
* 例化 hdmi_tx_reg_type 结构体，地址指向 CPU 总线基地址
*/
#define HDMI_TX_BASEADDR 0xF0111000
hdmi_tx_reg_type *hdmi_tx_reg = (hdmi_tx_reg_type*)HDMI_TX_BASEADDR;

int hdmi_tx_4k60_configure()
{
 hdmi_tx_parameter_type hdmi_tx_parameter;
 hdmi_tx_general_control_packet_type hdmi_tx_general_control_packet;
 hdmi_tx_avi_infoframe_packet_type hdmi_tx_avi_infoframe_packet;
 hdmi_tx_spd_infoframe_packet_type hdmi_tx_spd_infoframe_packet;
 hdmi_tx_audio_infoframe_packet_type hdmi_tx_audio_infoframe_packet;
 hdmi_tx_scdc_setting_type hdmi_tx_scdc_setting;

 anl_printf("HDMI2.0 Transmitter Configure Log\r\n");
 anl_printf("---------------\r\n");
 anl_printf("FPGA Version = %x\r\n",pl_reg_fpga_version_get());
 anl_printf("Reset HDMI2.0 Core....\r\n");
 anl_printf("Reset Serdes Phy....\r\n");

 anl_hdmi_tx_controller_reset_set(hdmi_tx_reg);
 anl_hdmi_tx_aux_packet_disable(hdmi_tx_reg);
 pl_reg_phy_serdes_reset_setting(1);

/*
* 配置 HDMI 4K FMC 子卡的重定时芯片 DP159 和时钟芯片
*/
 fmc_dp159_configure();
 fmc_si5319_148_5mhz_configure();

 anl_printf("Serdes Configure Done\r\n");
 pl_reg_phy_serdes_reset_setting(0);

```

```
/*
* 配置 HDMI2.0 Transmitter Core 的控制器参数，并写入到寄存器中
*/
 anl_printf("Initializing HDMI2.0 Core parameter....\r\n");
 hdmi_tx_parameter.htotal =4400;
 hdmi_tx_parameter.hsa   =88;
 hdmi_tx_parameter.hfp   =176;
 hdmi_tx_parameter.hbp   =296;
 hdmi_tx_parameter.hactive=3840;
 hdmi_tx_parameter.vtotal =2250;
 hdmi_tx_parameter.vsa   =10;
 hdmi_tx_parameter.vfp   =8;
 hdmi_tx_parameter.vbp   =72;
 hdmi_tx_parameter.vactive=2160;
 hdmi_tx_parameter.pixel_num=HDMI_TX_4_PIXEL;
 hdmi_tx_parameter.scramble_en=HDMI_TX_SCRAMBLE_ENABLE;
 hdmi_tx_parameter.tmds_clk_ratio=HDMI_TX_CLOCK_RATIO_1_40;
 hdmi_tx_parameter.vsync_polarity=HDMI_TX_VSYNC_HIGH_VALID;
 hdmi_tx_parameter.hsync_polarity=HDMI_TX_HSYNC_HIGH_VALID;
 hdmi_tx_parameter.over_sample=HDMI_TX_OVER_SAMPLE_DISABLE;
 hdmi_tx_parameter.pixel_color_depth=HDMI_TX_COLOR_DEPTH_24BIT;
 hdmi_tx_parameter.video_format=HDMI_TX_VIDEO_FORMAT_RGB;
 hdmi_tx_parameter.video_tpg_en=HDMI_TX_VIDEO_TPG_ENABLE;
 hdmi_tx_parameter.audio_channel_num=HDMI_TX_AUDIO_2_CHANNEL;

 anl_hdmi_tx_controller_parameter_updata(hdmi_tx_reg,&hdmi_tx_parameter);

/*
* 配置 HDMI Sink 端的 SCDC
*/
 anl_printf("Configure HDMI Sink SCDC....\r\n");
 hdmi_tx_scdc_setting.hdmi_tx_scdc_source_version=0x01;
 hdmi_tx_scdc_setting.scdc_scrambling=HDMI_TX_SCRAMBLING_ENABLE;
 hdmi_tx_scdc_setting.scdc_tmds_bit_clock_ratio=HDMI_TX_TMDS_CLOCK_RATIO_1_40;

 anl_hdmi_tx_scdc_setting_updata(hdmi_tx_reg,&hdmi_tx_scdc_setting);

 delay_ms(200);

 anl_printf("HDMI2.0 Core Reset Release\r\n");
 anl_hdmi_tx_controller_reset_release(hdmi_tx_reg);

 anl_printf("Initializing aux packet....\r\n");

```

```
80.  /*
81.  * 配置 General Control 数据包，并写入到寄存器中
82.  */
83.  hdmi_tx_general_control_packet.aux_paket_type = HDMI_TX_GENERAL_CONTROL_PACKET_TYPE;
84.  hdmi_tx_general_control_packet.color_depth = HDMI_TX_COLOR_DEPTH_24_BIT;
85.  hdmi_tx_general_control_packet.pixel_packing_phase = HDMI_TX_PACKING_PHASE_1OP4;
86.  anl_hdmi_tx_general_control_packet_updata(hdmi_tx_reg,&hdmi_tx_general_control_packet);
87.
88.  /*
89.  * 配置 AVI InfoFrame 数据包，并写入到寄存器中
90.  */
91.  hdmi_tx_avi_infoframe_packet.aux_paket_type = HDMI_TX_AVI_INFO_PACKET_TYPE;
92.  hdmi_tx_avi_infoframe_packet.version = 0x03;
93.  hdmi_tx_avi_infoframe_packet.length = 13;
94.  hdmi_tx_avi_infoframe_packet.colorspace = HDMI_TX_COLORSPACE_RGB;
95.  hdmi_tx_avi_infoframe_packet.active_format_information_present = HDMI_TX_NO_ACTIVE_FORMAT_INFORM
     ATION;
96.  hdmi_tx_avi_infoframe_packet.bar_data_present = HDMI_TX_BAR_DATA_NOT_PRESENT;
97.  hdmi_tx_avi_infoframe_packet.scan_infomation = HDMI_TX_SCAN_MODE_NONE;
98.  hdmi_tx_avi_infoframe_packet.colorimetry = HDMI_TX_COLORIMETRY_NONE;
99.  hdmi_tx_avi_infoframe_packet.picture_aspect_ratio = HDMI_TX_PICTURE_ASPECT_NONE;
100. hdmi_tx_avi_infoframe_packet.active_portion_aspect_ratio = HDMI_TX_ACTIVE_ASPECT_16_9_TOP;
101. hdmi_tx_avi_infoframe_packet.it_content = HDMI_TX_IT_IVALID;
102. hdmi_tx_avi_infoframe_packet.extended_colorimetry = HDMI_TX_EXTENDED_COLORIMETRY_XV_YCC_709;
103. hdmi_tx_avi_infoframe_packet.rgb_quantization_range = HDMI_TX_QUANTIZATION_RANGE_DEFAULT;
104. hdmi_tx_avi_infoframe_packet.non_uniform_picture_scaling = HDMI_TX_NUPS_UNKNOWN;
105. hdmi_tx_avi_infoframe_packet.video_code = 97;
106. hdmi_tx_avi_infoframe_packet.ycc_quantization_range = HDMI_TX_YCC_QUANTIZATION_RANGE_LIMITED;
107. hdmi_tx_avi_infoframe_packet.it_content_type = HDMI_TX_CONTENT_TYPE_GRAPHICS;
108. hdmi_tx_avi_infoframe_packet.pixel_repetition_factor = HDMI_TX_NO_REPETITION;
109. hdmi_tx_avi_infoframe_packet.end_of_top_bar = 0;
110. hdmi_tx_avi_infoframe_packet.start_of_bottom_bar = 0;
111. hdmi_tx_avi_infoframe_packet.end_of_left_bar = 0;
112. hdmi_tx_avi_infoframe_packet.start_of_right_bar = 0;
113. hdmi_tx_avi_infoframe_packet.additional_colorimetry_extension = HDMI_TX_SMPTE_ST_2113_P3D65;
114. anl_hdmi_tx_avi_infoframe_packet_updata(hdmi_tx_reg,&hdmi_tx_avi_infoframe_packet);
115.
116. /*
117.* 配置 SPD InfoFrame 数据包，并写入到寄存器中
118.*/
119. hdmi_tx_spd_infoframe_packet.aux_paket_type = HDMI_TX_SPD_INFO_PACKET_TYPE;
120. strcpy(hdmi_tx_spd_infoframe_packet.vendor_name_character,"anlogic");
121. strcpy(hdmi_tx_spd_infoframe_packet.product_description_character,"anlogic hdmi2.0");
122. hdmi_tx_spd_infoframe_packet.spd_source_information = HDMI_TX_SPD_SOURCE_PC_GENERAL;
```

```
123. anl_hdmi_tx_spd_infoframe_packet_updata(hdmi_tx_reg,&hdmi_tx_spd_infoframe_packet);
124.
125.  /*
126.* 配置 Audio InfoFrame 数据包，并写入到寄存器中
127.*/
128.hdmi_tx_audio_infoframe_packet.aux_paket_type = HDMI_TX_AUDIO_INFO_PACKET_TYPE;
129. hdmi_tx_audio_infoframe_packet.audio_coding_type = HDMI_TX_AUDIO_CODING_TYPE_STREAM;
130. hdmi_tx_audio_infoframe_packet.audio_channel_number = HDMI_TX_AUDIO_INFO_2_CHANNELS;
131. hdmi_tx_audio_infoframe_packet.audio_sample_frequency = HDMI_TX_AUDIO_SAMPLE_FREQUENCY_STREAM;
132. hdmi_tx_audio_infoframe_packet.audio_sample_bits = HDMI_TX_AUDIO_SAMPLE_BITS_STREAM;
133. anl_hdmi_tx_audio_infoframe_packet_updata(hdmi_tx_reg,&hdmi_tx_audio_infoframe_packet);
134.
135. anl_printf("Enable HDMI2.0 Core Aux Packet Sending\r\n");
136. anl_hdmi_tx_aux_packet_enable(hdmi_tx_reg);
137.
138. anl_printf("\r\n");
139. anl_printf("Printf Aux Packet:\r\n");
140. anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg,HDMI_TX_GENERAL_CONTROL_PACKET_TYPE);
141. anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg,HDMI_TX_AVI_INFO_PACKET_TYPE);
142. anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg,HDMI_TX_SPD_INFO_PACKET_TYPE);
143. anl_hdmi_tx_aux_packet_read_printf(hdmi_tx_reg,HDMI_TX_AUDIO_INFO_PACKET_TYPE);
144.
145. anl_printf("\r\n");
146. anl_hdmi_tx_edid_read_printf(hdmi_tx_reg);
147. anl_hdmi_tx_scdc_read_printf(hdmi_tx_reg);
148.
149. return 0;
150.}
```

HDMI2.0 Transmitter 启动的打印日志如图 5-3 所示。

```
HDMI2.0 Transmitter Configure Log
Configure Resolution 3840*2160 60Hz
---------------
FPGA Version = 24093001
Reset HDMI2.0 Core....
Reset Serdes Phy....
Initializing Hdmi 4k Fmc Card DP159....
Initializing Hdmi 4k Fmc Card DP159 Successful
Initializing Hdmi 4k Fmc Card Si5319....
Configure Si5319 output clock frequency 148.5MHz....
Wating for Si5319 PLL Locked....
Si5319 PLL Has Locked
Initializing Hdmi 4k Fmc Card Si5319 Successful
Serdes Configure Done
Initializing HDMI2.0 Core parameter....
Configure HDMI Sink SCDC....
HDMI2.0 Core Reset Release
Initializing aux packet....
Enable HDMI2.0 Core Aux Packet Sending

Printf Aux Packet:
general_control_packet: 03 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
avi_infoframe_packet: 82 03 0D FB 00 02 10 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
spd_infoframe_packet: 83 01 19 0E 61 6E 6C 6F 67 69 63 20 61 6E 6C 6F 67 69 63 20 68 64 6D 69 32 2E 30 20 09 00 00
audio_infoframe_packet: 84 01 0A 70 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

read hdmi sink edid data:
00h: 00 FF FF FF FF FF FF 00 05 E3 03 32 D6 02 00 00
01h: 27 20 01 03 80 46 27 78 2E 20 75 AE 50 49 A8 27
02h: 0A 50 54 BF EF 00 D1 C0 B3 00 95 00 81 80 81 40
03h: 81 C0 01 01 01 01 4D D0 00 A0 F0 70 3E 80 30 40
04h: 35 00 B9 88 21 00 00 1A 00 00 00 FF 00 54 4B 41
05h: 4E 39 4A 41 30 30 30 37 32 36 00 00 00 FC 00 55
06h: 33 32 4E 33 47 36 52 33 42 0A 20 20 00 00 00 FD
07h: 00 28 3C 1E 8C 3C 00 0A 20 20 20 20 20 20 01 59
08h: 02 03 47 F1 4C 90 04 03 1F 13 01 12 5D 5E 5F 60
09h: 61 23 09 07 07 83 01 00 00 6D 03 0C 00 10 00 38
0Ah: 3C 20 00 60 01 02 03 67 D8 5D C4 01 78 80 03 E3
0Bh: 05 E3 01 E3 0F 00 0C E6 06 07 01 63 63 00 68 1A
0Ch: 00 00 01 01 28 3C E6 A3 66 00 A0 F0 70 1F 80 30
0Dh: 20 35 00 B9 88 21 00 00 1A 56 5E 00 A0 A0 A0 29
0Eh: 50 30 20 35 00 B9 88 21 00 00 1E 4D 6C 80 A0 70
0Fh: 70 3E 80 30 20 3A 00 B9 88 21 00 00 1A 00 00 96

read hdmi sink scdc data:
00h: 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00
01h: 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02h: 03 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04h: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
05h: 93 81 28 82 92 82 01 00 00 00 00 00 00 00 00 00
06h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Ah: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Bh: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Ch: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Dh: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Eh: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0Fh: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图 5-3 HDMI2.0 Transmitter 启动日志

6 开发环境

HDMI2.0 Transmitter 在 AP106_V2.0 开发板上进行开发验证，整个开发环境如图 6-1 所示。**由于 SerDes 输出的电平标准不适用于 HDMI 电平，因此需要 HDMI 重定时器对 SerDes 输出的信号进行转换和增强**，这里使用了 MLK-V-FMC3001 4K HDMI 子卡来实现，MLK-V-FMC3001 子卡如图 6-2 所示。

HDMI2.0 Transmitter 测试工程中内置了多个分辨率，4Kp60、4Kp30、1080p60，可以通过板上的按键动态切换。按键位置见图 6-1 标识。测试工程内部还设置了 I2S 接收器，用于接收 I2S 音频数据测试音频功能，输入的音频采样率需要固定为 48K，I2S 的信号接口通过 J8 座子输入，信号位置见图 6-1 标识。

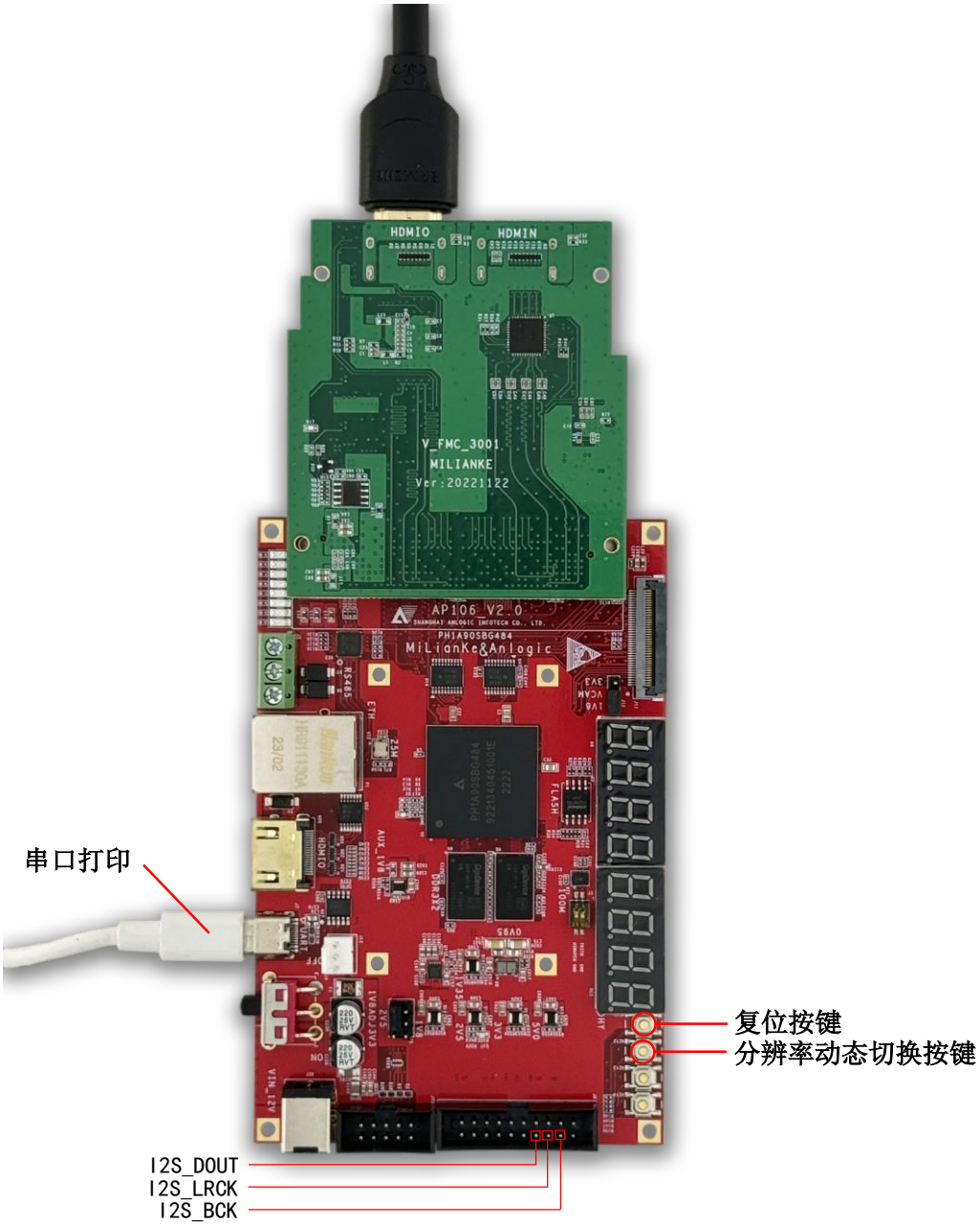

图 6-1 HDMI2.0 Transmitter 开发环境

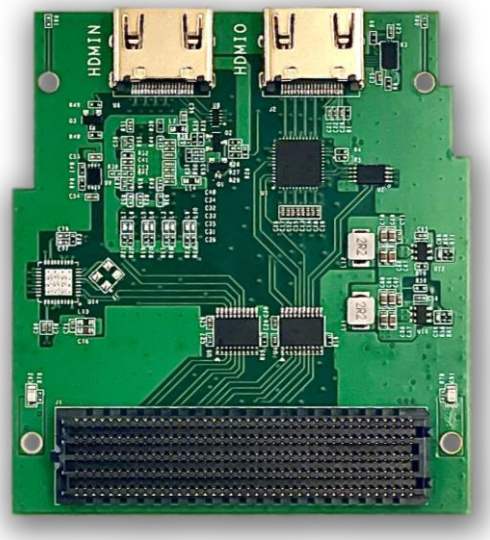

图 6-2 MLK-V-FMC3001 子卡

7 资源消耗

HDMI2.0 Transmitter 资源消耗如表 7-1 所示。

表 7-1 HDMI2.0 Transmitter 资源消耗

LUT6	REG	ERAM	SerDes Lane
7709	10987	17	4

8 工程文件信息

本示例工程提供如下文件。

表 8-1 文件信息

参数	说明
Reference Design	Yes
RTL Language	Verilog
Test bench	N/A
Test bench Format	N/A
Simulation	N/A
C	Yes
IP Model	N/A
Project Platform	N/A
TD Soft Version	TD5.6.88061

9 参考文档

[1] High-Definition Multimedia Interface Specification Version 1.4b

[2] High-Definition Multimedia Interface Specification Version 2.0

[3] VESA Enhanced Extended Display Identification Data Standard 1.4

[4] ANSI/CTA-861-H

[5]IEC 60958-1

[6]IEC 60958-3

版本信息

日期	版本	修订记录
2024/9/30	1.0	1.名称修改为 HDMI2.0 Transmitter 2.新增 HDMI2.0 协议中的加扰功能 3.新增 HDMI2.0 协议中的 SCDC 功能 4.新增 CPU 控制总线，HDMI 协议层的配置流程采用 CPU 配置 5.新增音频时钟恢复辅助数据包（ACR）硬件发送端口 6.修改了音频发送接口，兼容性和灵活性更强 7.新增了对于新器件的支持，包括 PH2A 系列、PH1P 系列、DR1 系列 8.新增了 CPU 驱动，采用 CPU 配置 HDMI 所有的工作流程 9.修改整体架构，协议层和 serdes 物理层相互分离

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其他方式授予任何知识产权许可；本文档仅为向用户提供使用器件的参考，协助用户正确地使用安路科技产品之用，其著作权归安路科技所有；本文档所展示的任何产品信息均不构成安路科技对所涉产品或服务作出任何明示或默示的声明或保证。

安路科技将不定期地对本文档进行更新、修订。用户如需获取最新版本的文档，可通过安路科技的官方网站（网址为：https://www.anlogic.com）自行查询下载，也可联系安路科技的销售人员咨询获取。