

1 OV5640 摄像头采集 IO_HDMI 显示屏显示设计

工程源码	02_设计实例 -- acz702_ov5640_ddr3_hdmi_VGA.zip
相关视频课程	
	如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

章节导读

通过前面章节，我们已经使用 OV5640 摄像头完成了 TFT 屏、VGA 显示器、基于 SiI9022 的 HDMI 显示器的显示系统设计。从系统知识的完整性上来说，除了通过 HDMI 芯片完成 HDMI 显示外，我们同样能通过 IO 模拟的方式实现 HDMI 显示。

为此，本章将使用 ACM_VGAHDMI 模块，带大家通过 FPGA 内部模拟的方式完成 OV5640 图像采集的 HDMI 显示设计。

1.1 VGA 与 HDMI 显示对比

虽然使用 VGA 显示器已经能够实现较高分辨率的图像显示，但是 VGA 显示有个致命缺点，那就是在显示高分辨率的图像时，可能会出现失真、图像出现虚影的现象。而 HDMI 显示同样能够支持较高分辨率，甚至所能支持的最大分辨率要高于 VGA，但是 HDMI 在进行高分辨率显示时，极少甚至不会出现失真、虚影等现象。而这一切则是因为这两种显示设备接口在传输数据时，使用的是不同类型数据。图 1-1 为现今常见的 VGA 显示器和 HDMI 显示器与显卡传输图像数据的数据流变换示意图：

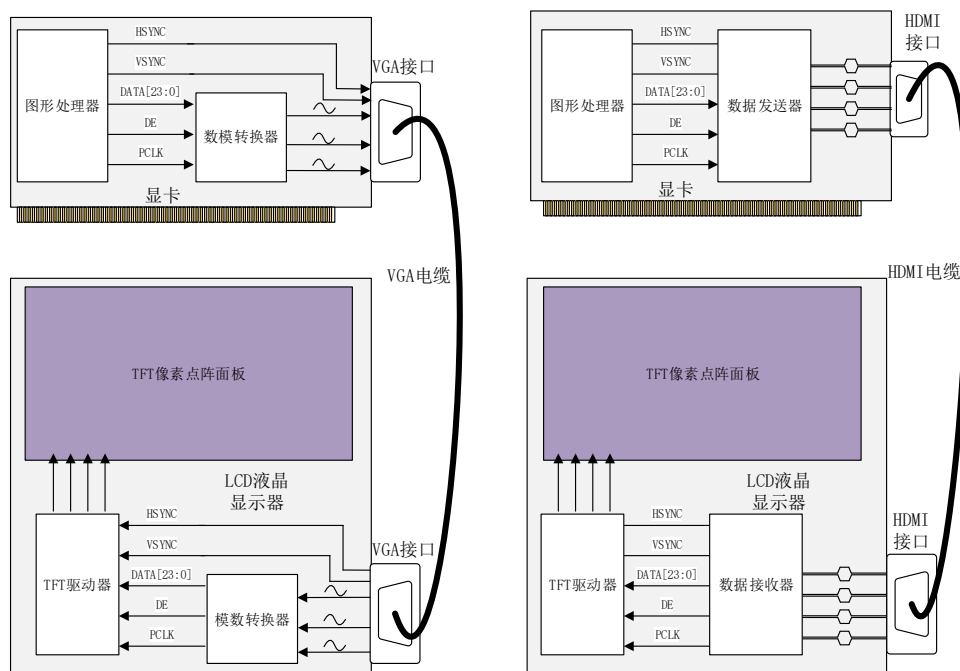


图 1-1 VGA 和 HDMI 接口与显卡传输图像数据流变换图

现今常见的无论是 VGA 接口显示器还是 HDMI 接口显示器，大多使用的液晶屏，这类屏幕采用的是液晶材质，通过将众多细小的液晶颗粒按照矩阵的形式排布在一起，实现显示面板。这些液晶会根据加载其两端的电压大小而改变透光性，进而实现被动的颜色显示。

对于一个液晶屏而言，其所能显示的物理像素数量是确定的，每个像素点的颜色都可以对应一个图像数据，而这个数据是数字信号，所以液晶显示屏从原理上讲可以认为是数字显示屏（虽然最终控制单颗液晶的透光程度时也会将这个数字信号转换为模拟电压信号，但是这已经是像素点级别的成像原理了，而非液晶显示器显示整幅图案的成像原理）。显示时，只需要得到对应像素点的颜色数据即可，所以这类显示屏接收的是数字信号。

而 VGA 接口传输的是模拟信号，因此显卡输出的数字图像信号必须转换成模拟信号后才能通过 VGA 接口输出到 VGA 显示器。而图像信号在送到 VGA 显示器的 VGA 接口后还需要再经过模数转换，转换为数字信号后才能驱动液晶显示。

数据在模数转换和数模转换的过程容易发生变化，从而导致显示的图像存在一些细微的差别。如果在转化时数据变化速度过快，就可能造成拖影现象。并且由于 VGA 线缆传输的是模拟信号，模拟信号在传输过程中容易受到噪声的干扰，导致显示的图像中会出现很多杂点。因此在较高分辨率的图像显示时，VGA 模拟信号传输的这些缺点就越发的明显了。

而本章也将基于“OV5640 摄像头采集 VGA 显示屏显示”实验，通过实现 1280*720@60Hz 图像的 HDMI 显示，带领大家学习如何将 OV5640 采集到的图像数据，通过 IO 模拟的 HDMI 接口的方式，将图像数据显示到 HDMI 显示器上。

1.2 基于 OV5640 的 HDMI 显示屏显示系统

下图 1-2 为本次设计的系统框图:

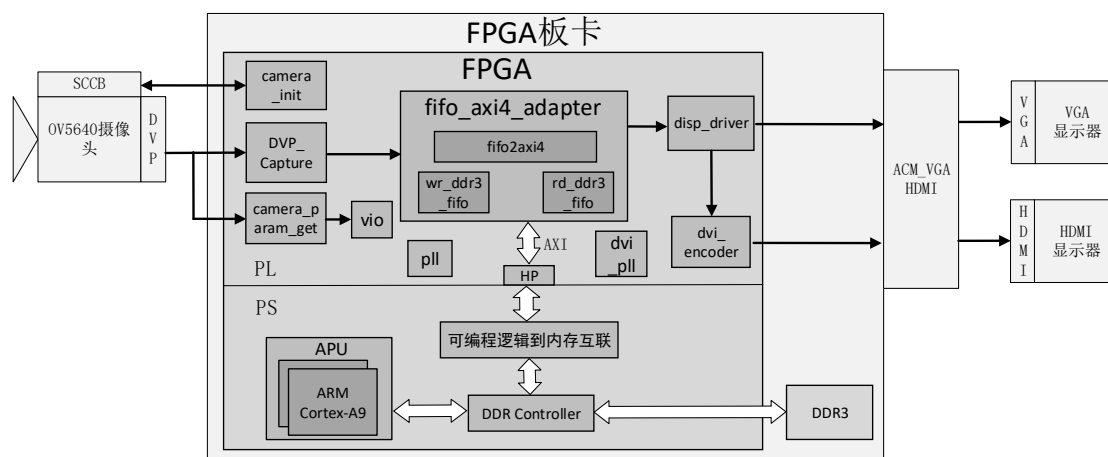


图 1-2 系统框图

本次设计基于“OV5640 摄像头采集 VGA 显示屏显示”实验，对于设计来说，要想实现 HDMI 显示，只需要对 disp_driver 模块输出的 1280*720@30Hz 图像数字信号以及对应行场同步信号进行编码，转换为符合 HDMI 接口的编码格式即可。因此在设计中添加了以下模块：

1. `dvi_pll` (PLL 核), 用于产生 `dvi` 编码模块工作所需的时钟
2. `dvi` 编码模块 (`dvi_encoder`), 用于将 `disp_driver` 模块输出的图像数据与时序, 转换为符合 `HDMI` 接口发送时序的编码格式。

对于 HDMI 显示来说，从 `disp_driver` 模块输出的图像数据以及时序信号需要经由 `dvi_encoder` 核进行 TMDS 编码，并转换为串行信号后才能通过 HDMI 接

口输出。输出的信号在通过 HDMI 线缆传输给 HDMI 显示器的 HDMI 接口后，显示器内部的 TMDS 解码模块会对信号解码，然后进行串并转换，将信号转变为并行信号后依据时序信号，将图像显示出来，实现 1280*720@60Hz 的 HDMI 显示。

1.3 产生 DVI 驱动时钟

对于 dvi_encoder 模块来说，所需的时钟有两个，一个是进行 TMDS 编码的时钟，一个是编码后对信号进行并串转换的时钟。

dvi_encoder 模块会对接收的 RGB888 数据和行场同步信号进行 TMDS 编码，这些信号会被分成三组，每组由两位控制信号和对应的 8 位颜色分量组成（其中两组的控制信号为空）也就是每组一共 10 位数据。在设计中这些信号由 disp_driver 提供，因此 TMDS 编码时使用的也是 disp_driver 输出的工作时钟，即 74.25MHz。

而在进行并串转换时，dvi_encoder 模块会在每个时钟的上升沿和下降沿对每组信号进行并行到串行的转换，为了能够将编码完成的数据立马转换，就需要一个 5 倍于 TMDS 编码时的工作时钟，也就是 $74.25 \times 5 = 371.25\text{MHz}$ 的时钟。

为了保持数据的同步性，我们就需要保证这两个时钟的同源，而如果直接通过已有的 PLL 生成，实际只能产生 73.684MHz 和 350MHz 的时钟，如图 1-3 所示：

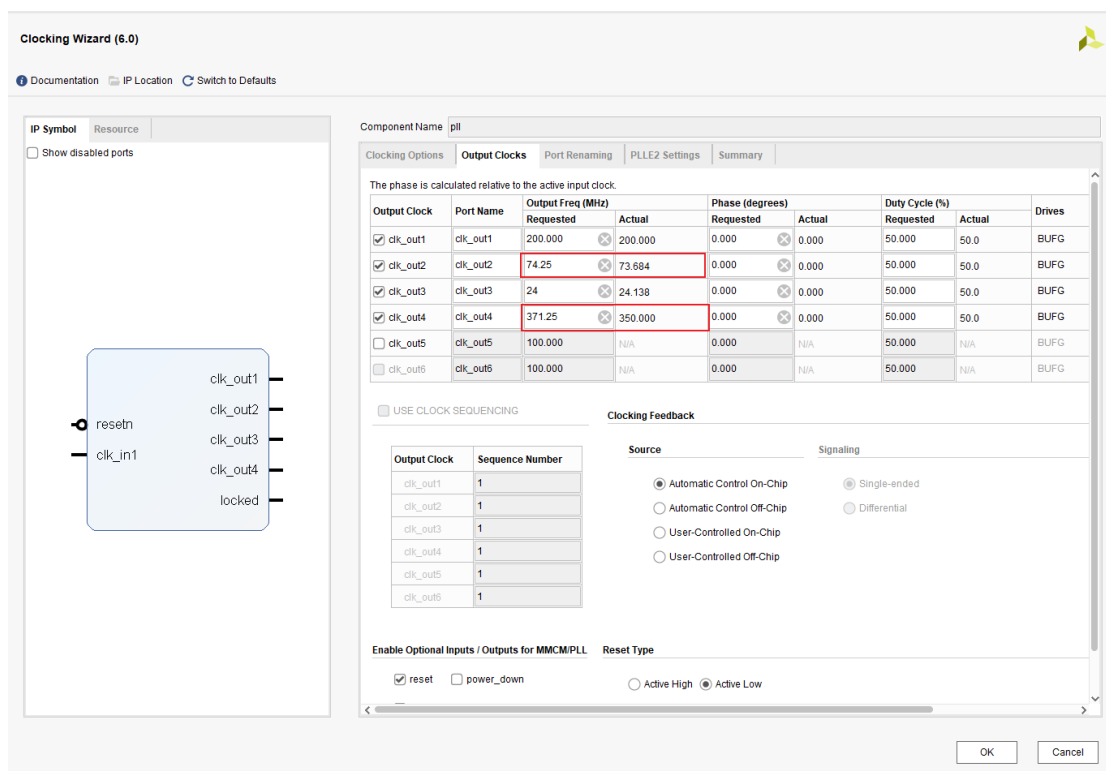


图 1-3 生成时钟（错误示范）

这是因为 PLL 首先会根据所需时钟将输入时钟倍频到一个较高值，然后再针对每个输出时钟分频，尽量去满足每个时钟需求，而时钟越多，就越难保证每个时钟都能满足需求。

简单计算我们就可以知道这两个时钟并不能满足 5 倍的关系，因此这里我们可以产生一个 100MHz 的时钟，随后新建一个 PLL 核，命名为 dvi_pll 方便区分，将 100MHz 时钟作为 dvi_pll 的输入，输出所需的 74.25MHz 时钟和 371.25MHz 时钟。由于 dvi_pll 使用的是 pll 输出的时钟，所以我们需要将 dvi_pll 的输入时钟源设置为 Global buffer 或者 No buffer。相关配置分别如图 1-4、图 1-5 和图 1-6 所示：

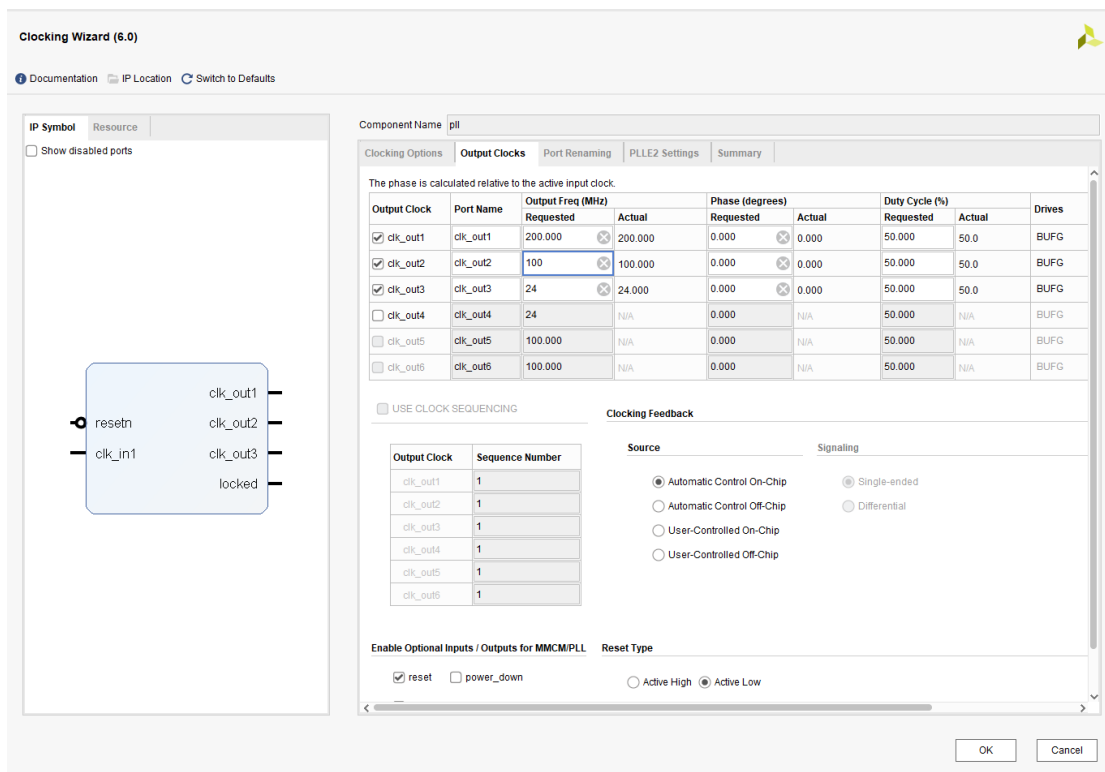


图 1-4 PLL 核

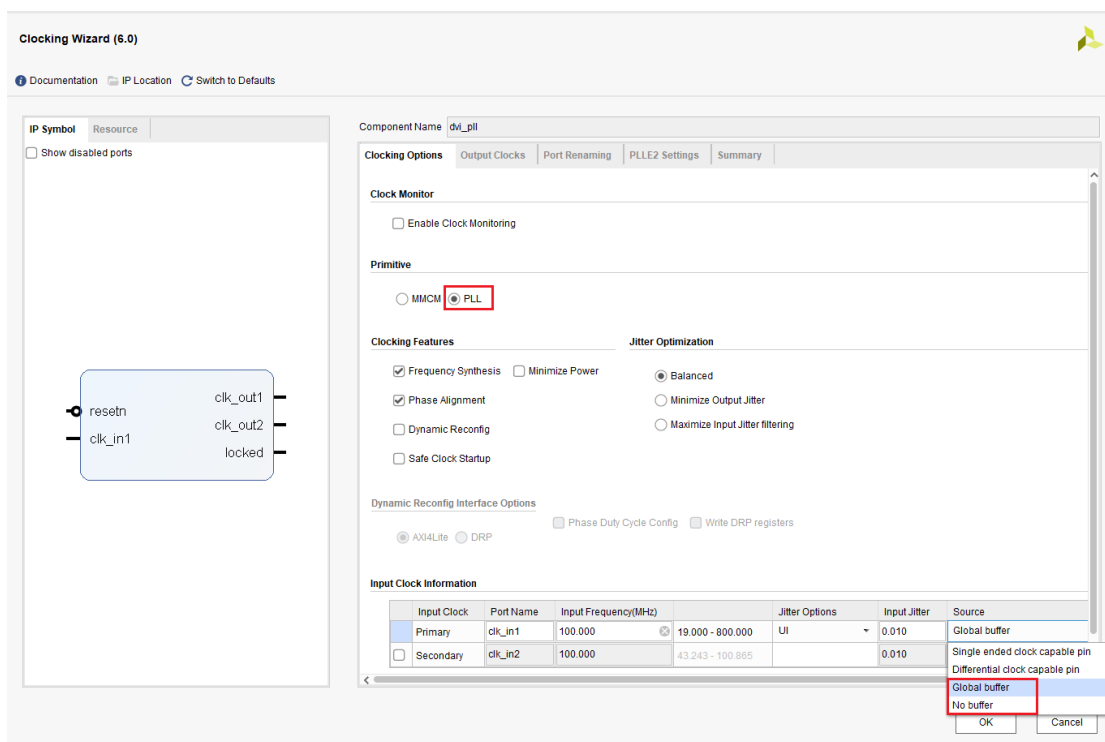


图 1-5 设置 dvi_pll 时钟源

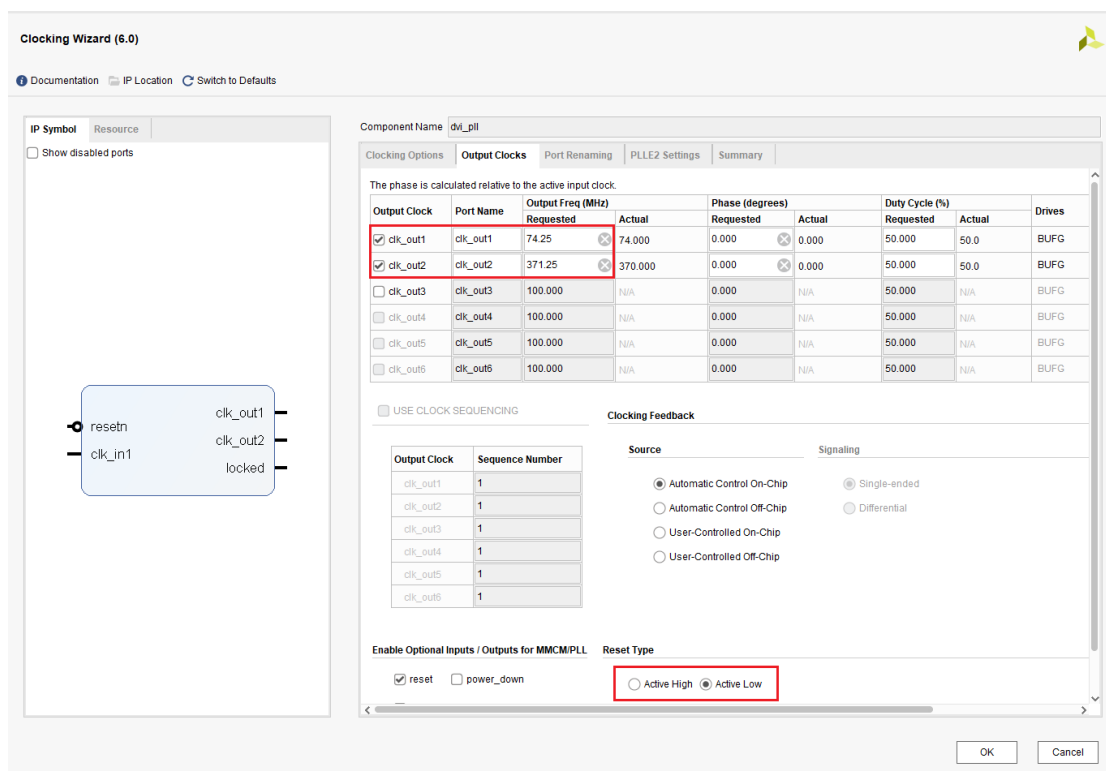


图 1-6 dvi_pll

如果此时，PLL 还是无法产生 74.25M 和 371.25M 时钟，可以将图 1-5 中 Primitive 设置为 MMCM。同时，新建 PLL 的复位类型需要手动设置为低电平有效。配置完成两个 PLL 后便需要在顶层中修改 pll 的例化并添加 dvi_pll 的例化，相关代码如下：

```

wire      loc_clk100m;
wire      disp_clk;
wire      disp_clk5x;

pll pll
(
    // Clock out ports
    .clk_out1 (loc_clk200m      ), // output clk_out1
    .clk_out2 (loc_clk100m      ), // output clk_out2
    .clk_out3 (loc_clk24m       ), // output clk_out3
    // Status and control signals
    .resetn   (pl_reset_n      ), // input reset
    .locked   (pll_locked       ), // output locked
    // Clock in ports
    .clk_in1  (ps2pl_clk50m_0   ) // input clk_in1
);

dvi_pll instance_name
(

```

```
// Clock out ports
.clk_out1(dispc_clk),      // output clk_out1
.clk_out2(dispc_clk5x),    // output clk_out2
// Status and control signals
.resetn(!reset), // input resetn
.locked(),          // output locked
// Clock in ports
.clk_in1(loc_clk100m)
);
```

例化完成后，我们便完成了 dvi_encoder 模块时钟的生成，接下来就是 dvi_encoder 模块的设计和例化工作了。

1.4 DVI 编码模块

DVI 编码模块负责对信号进行 TMDS 编码以及并行到串行的转换，对于该模块的详细描述，可以参考“[错误!未找到引用源。](#)”章节。这里我们直接使用设计好的模块即可，使用时只需将该模块以及相关子模块（可以直接从例程中提取）添加进工程中，在顶层中添加相关输出接口并例化 DVI 编码模块。

顶层中对 dvi_encoder 模块接口的添加以及模块例化代码如下：

```
//添加接口
//hdmi interface
output      hdmi_clk_p   ,
output      hdmi_clk_n   ,
output [2:0] hdmi_dat_p   ,
output [2:0] hdmi_dat_n   ,

//HDMI
dvi_encoder dvi_encoder1(
    .pixelclk    (dispc_clk ),
    .pixelclk5x  (dispc_clk5x ),
    .rst_p       (reset      ),
    .blue_din    (VGA_rgb[7:0] ),
    .green_din   (VGA_rgb[15:8] ),
    .red_din     (VGA_rgb[23:16] ),
    .hsync       (VGA_hs      ),
    .vsync       (VGA_vs      ),
    .de          (VGA_de      ),
    .tmads_clk_p (hdmi_clk_p ),
    .tmads_clk_n (hdmi_clk_n ),
    .tmads_data_p (hdmi_dat_p ),
    .tmads_data_n (hdmi_dat_n )
);
```

至此我们便完成了所有相关模块的设计，由于设计中我们在顶层中添加了

1.5 引脚分配

表 1-1 HDMI 接口引脚分配表

Pin Name	Signal Name	Pin NO.		Pin Name	Signal Name	Pin NO.
HDMI_CLK_P	hdmi_clk_p	N20		HDMI_D1_P	hdmi_dat_p[1]	L19
HDMI_CLK_N	hdmi_clk_n	P20		HDMI_D1_N	hdmi_dat_n[1]	L20
HDMI_D0_P	hdmi_dat_p[0]}	M19		HDMI_D2_P	hdmi_dat_p[2]	K16
HDMI_D0_N	hdmi_dat_n[0]}	M20		HDMI_D2_N	hdmi_dat_n[2]	J16

图 1-7 所示:

IO Ports													
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	IO Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	
All ports (184)													
DDR_38849 (71)	(Multiple)			✓		502 (Multiple)*	(Multiple)	(Multiple)	(Multiple)		NONE	FP_VTT_50	
FIFO_READ_3567 (1)	OUT			✓		35 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
FIXED_IO_38849 (59)	INOUT			✓		(Multiple) (Multiple)*	(Multiple)	(Multiple)			NONE	FP_VTT_50	
read_clk_3567 (1)	IN			✓		35 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
write_clk_18657 (1)	IN			✓		35 LVCMOS33*	3.300				NONE	NONE	
camera_data (8)	IN			✓		35 LVCMOS33*	3.300				NONE	NONE	
hdmi_dat_p (5)	OUT	hdmi_dat_n		✓		35 TMDS_33*	3.300				NONE	FP_3_3_50	
hdmi_dat_p[2]	OUT	hdmi_dat_n[2]	K16	✓		35 TMDS_33*	3.300				NONE	FP_3_3_50	
hdmi_dat_p[1]	OUT	hdmi_dat_n[1]	L19	✓		35 TMDS_33*	3.300				NONE	FP_3_3_50	
hdmi_dat_p[0]	OUT	hdmi_dat_n[0]	M19	✓		35 TMDS_33*	3.300				NONE	FP_3_3_50	
VGA_rgb (24)	OUT			✓		(Multiple) LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
Scalar ports (13)													
camera_href	IN		U13	✓		34 LVCMOS33*	3.300				NONE	NONE	
camera_sclk	OUT		N17	✓		34 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
camera_sdat	INOUT		P18	✓		34 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
camera_vsync	IN		N16	✓		35 LVCMOS33*	3.300				NONE	NONE	
camera_xclk	OUT		A20	✓		35 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
hdmi_clk_p	OUT	hdmi_clk_n	N20	✓		34 TMDS_33*	3.300				NONE	FP_3_3_50	
led	OUT		T14	✓		34 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
reset_n	IN		F20	✓		35 LVCMOS33*	3.300				NONE	NONE	
Si9022_sclk	OUT		T10	✓		34 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
Si9022_sdat	INOUT		R14	✓		34 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
hdmi_hs	OUT		L15	✓		35 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	
VGA_vs	OUT		K14	✓		35 LVCMOS33*	3.300		12	✓	SLOW	FP_VTT_50	

图 1-7 HDMI 差分引脚约束

描述文件导出到 SDK，就可以开始烧录验证了。

1.6 板级验证

显示”实验的验证。

1.6.1 所需硬件

1. ACZ702 开发板 x1
2. OV5640 摄像头 x1
3. ACM_VGAHDMI 模块 x1
4. HDMI 显示器 x1
5. 电源线 x1（可选）
6. type-c 下载线 x1
7. HDMI 线缆 x1

1.6.2 硬件连接

本次设计的硬件连接，如图 1-8 所示：

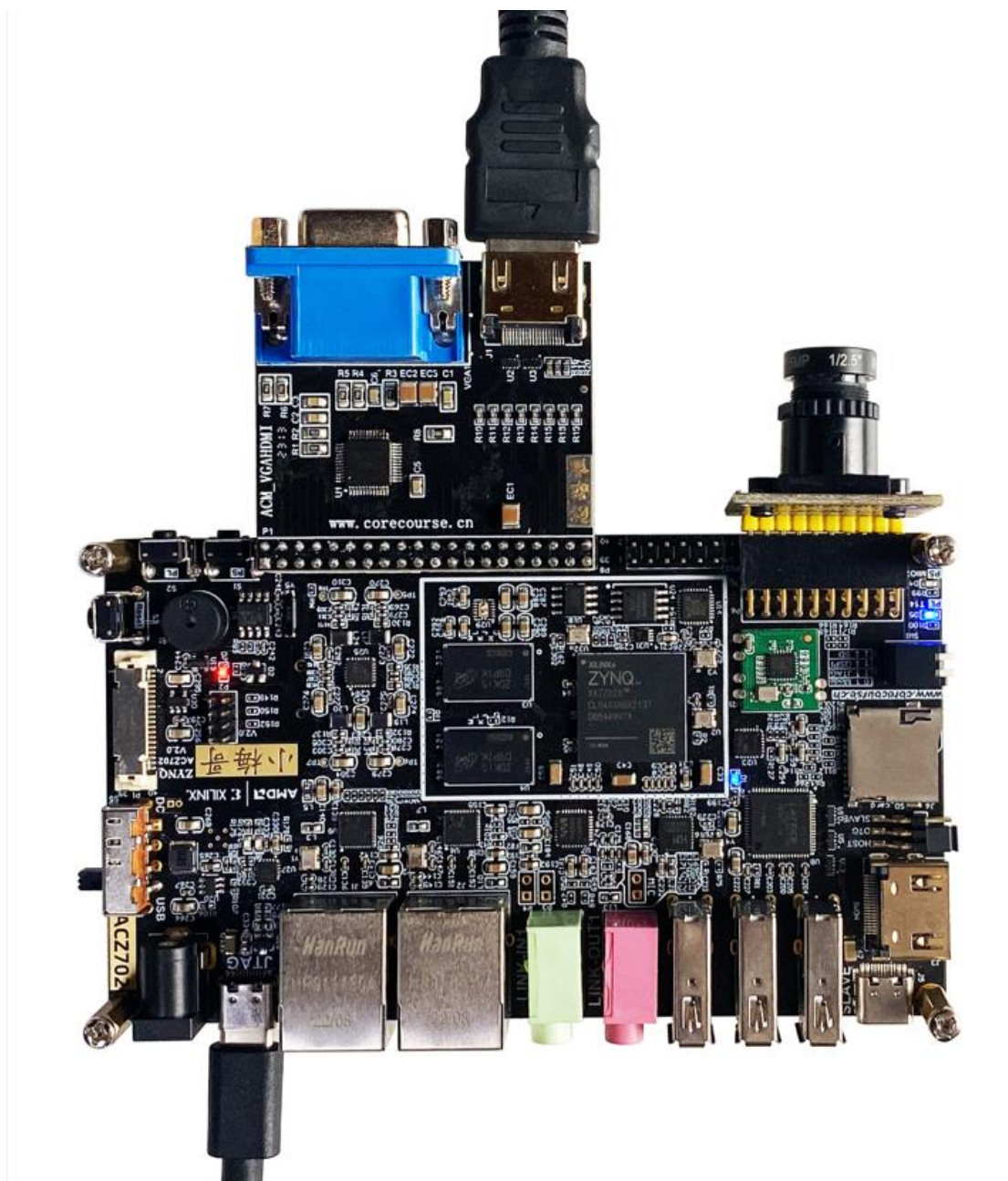


图 1-8 硬件连接

ACM_VGAHDMI 模块通过开发板上的 40pin 拓展接口与开发板相连，正确连接时，所有针脚都应该被插接上

在连接 HDMI 接口时，需要注意：**HDMI 线缆必须一端连接 ACM_VGAHDMI 模块的 HDMI 接口，另一端连接显示器或电视机的 HDMI 接口。**

不可以将线缆一端连接 ACM_VGAHDMI 模块的 HDMI 接口，另一端连接到笔记本或者主机的 HDMI 接口上。因为笔记本和主机的 HDMI 接口为输出接

口，而设计中 ACM_VGAHDMI 模块的 HDMI 接口用于输出，这些输出接口互相连接无法实现图像的显示。而显示器和电视机上的 HDMI 接口为输入接口，HDMI 输出接口只有与这些输入接口相连，才能实现图像的显示。

连接完毕后拨动开关为开发板上电，通过 SDK 将设计烧录到开发板中。系统开始工作，实物场景、显示效果以及整体场景分别如图 1-9、图 1-10 和图 1-11 所示：

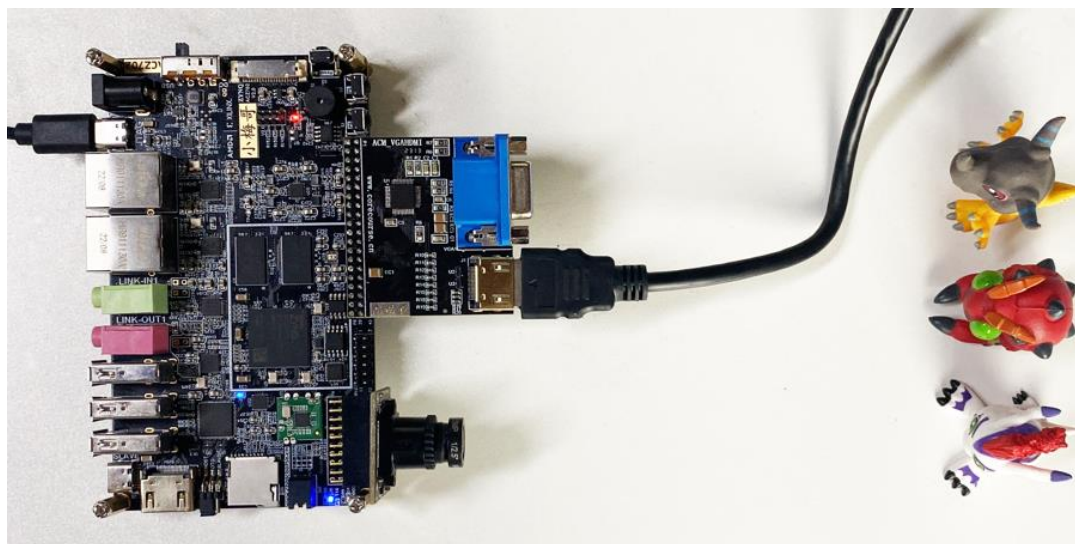


图 1-9 实物场景

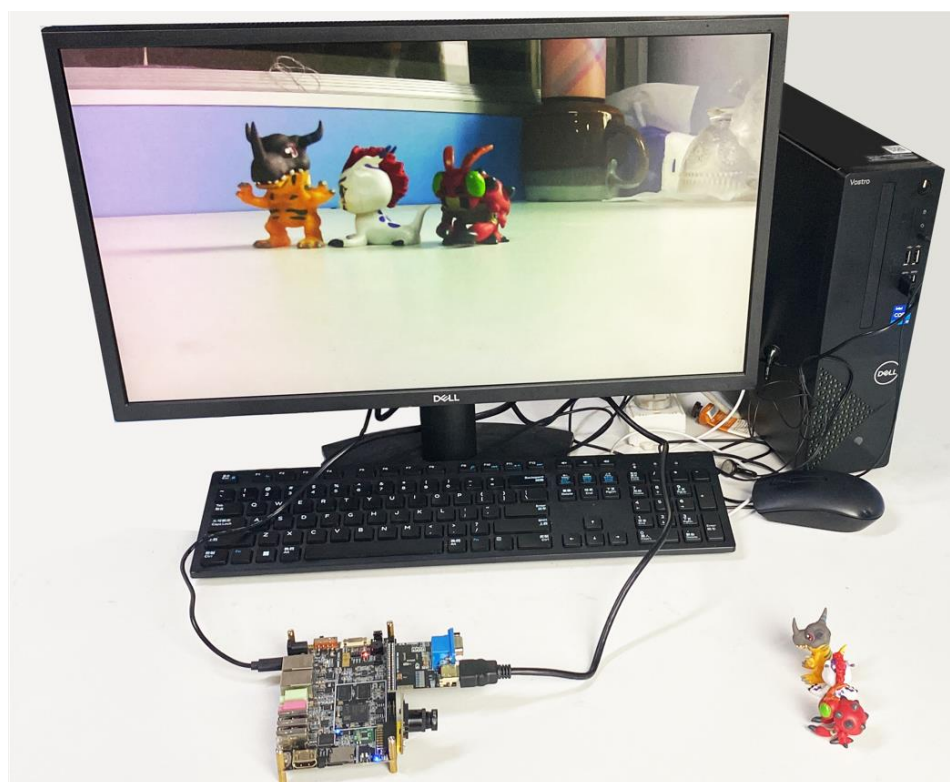


图 1-10 整体显示效果

可以看到图像显示正常，没有出现任何撕裂重叠现象，颜色层次清晰。接下来回到 Vivado 中，连接硬件并将 ltx 文件烧录进开发板中，VIO 中接收到的数据如下：



hw_vio_1					
<div><div>Q</div><div>≡</div><div>⬆</div><div>+</div><div>-</div></div>					
Name	Value	Acti...	Directi...	VIO	
>  Fpclk[31:0]	[U] 84000000		Input	hw_vio_1	
>  Fps[7:0]	[U] 30		Input	hw_vio_1	

图 1-11 VIO 显示界面

可以看到，摄像头输出的像素时钟与帧率都与预期一致，且十分稳定，说明整个设计是稳定且正常工作的，本次设计成功。

1.7 总结

本章带大家实现了 OV5640 摄像头采集 HDMI 显示屏显示系统的设计与验证，设计整体上只是在“OV5640 摄像头采集 VGA 显示屏显示”系统的基础上对信号进行了 TMDS 编码和并串转化，从而以符合 HDMI 编码的格式输出。