

1 OV5640 摄像头采集 VGA 显示屏显示设计

工程源码	02_设计实例 -- acz702_ov5640_ddr3_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

通过上一章节的学习，我们完成了 OV5640 图像采集的 TFT 和 HDMI 显示系统的设计，但是市面上的 TFT 屏大都只能支持一些低分辨率的图像显示。例如上一章中我们使用的 5 寸屏，其分辨率为 800*480，因此最高就只能显示 800*480 分辨率的图像。受 TFT 屏本身分辨率的影响，这个系统通常只能被应用于一些低分辨率图像显示的场景。

而 FPGA 除了能够驱动 TFT 屏外，还能够驱动我们的电脑显示器，例如 VGA 接口显示器、HDMI 接口显示器等等。因而对于一些高分辨率的图像显示设计，开发者大都倾向于使用能够提供更高分辨率的电脑显示器，本章将以“OV5640 图像采集的 TFT 屏显示系统”为基础，带大家学习如何以较高的分辨率将 OV5640 的图像采集并显示在 VGA 显示器上。

1.1 基于 OV5640 的 VGA 显示屏显示系统

常见的高分辨率有很多，例如 720p(1280*720)、1080p(1920*1080)、2k(2560*1440)、4k(4096*2160)等，考虑到 OV5640 输出图像的帧率会随着分辨率的增高而变小。例如，在输出 640*480 分辨率图像时可以达到 90 帧，而输出 2591*1944 分辨率图像时只能达到 15 帧。本章将通过摄像头产生 1280*720 的图像数据，在“OV5640 图像采集的 TFT 屏显示系统”的基础上，实现数据在 VGA 显示屏上 1280*720 分辨率的显示设计。

首先是本次设计的系统框图，如图 1-1 所示：

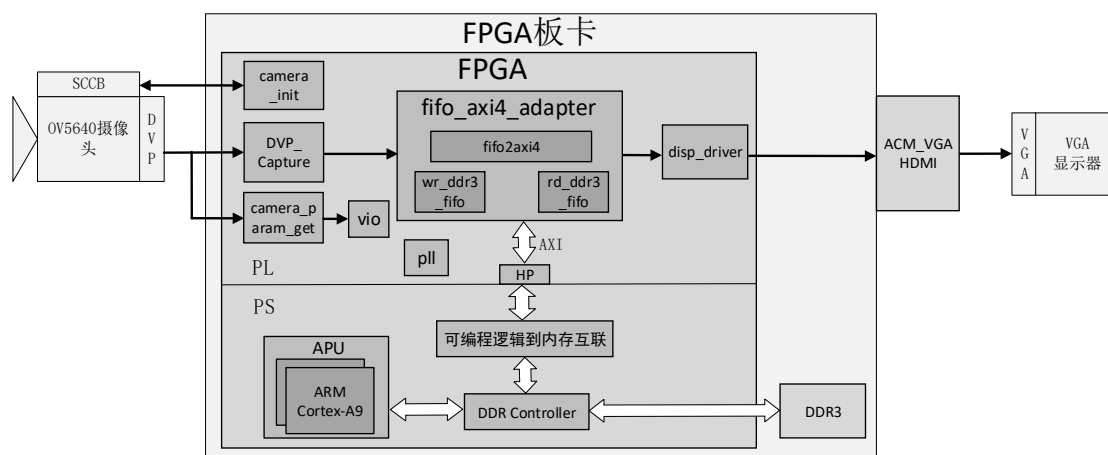


图 1-1 系统框图

本次设计基于“OV5640 的图像采集显示系统”（对应工程为 acz702_ov5640_ddr3_tft_hdmi），在该系统上进行了一定的添加与修改，具体如下：

1. 添加了一个 camera_param_get 模块，用来确认摄像头输出的图像帧率，除此之外该模块还能测量 pclk 的频率，这些数据最终通过 vio 显示
2. 将输出显示修改成 VGA 显示
3. 对部分模块参数进行了修改。

在讲解 RGB 接口 TFT 屏扫描方式时，我们曾讲过，RGB 接口的 TFT 屏其扫描方式与 VGA 一致，因此，TFT 驱动模块除了用来驱动 TFT 屏，也能用来驱动 VGA 显示屏。但是，值得注意的一点是，TFT 屏直接使用的数字信号，而 VGA 使用的是模拟信号。因此，在驱动模块输出图像时序和数据后，还需要使用专门的硬件将这些数字信号转变为模拟信号。这类转换器，最常用的芯片就是 ADI 公司的 ADV7123。

为了实现 VGA 显示器的驱动，芯路恒公司提供了一个扩展模块 ACM_VGAHDMI。该模块使用兼容 ADI 公司的 GM7123 芯片，芯片内部包含 3 个 10 位的 DAC。在设计时，模块仅使用了芯片内每个 DAC 的高 8 位，因此支持 24 位数字像素输入，输入后的数据经过 DAC 转换为模拟量，并支持高达 1920*1080 的 VGA 图像输出。ACM_VGAHDMI 模块如下图 1-2：

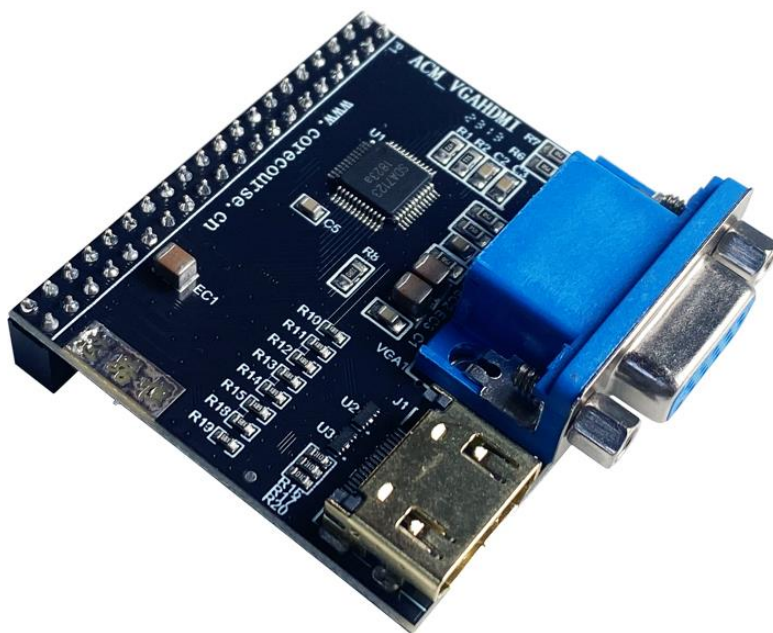


图 1-2 ACM_VGAHDMI 输出模块

由于一个 24 位色的 VGA 电路至少需要占用 28 个 I/O，直接集成在开发板上会浪费较多的 I/O，所以在 ACZ702 开发板上我们通过 40pin 拓展接口连接该输出模块，来实现 VGA 输出显示。

为了方便理解设计，接下来我们按照数据流向对设计模块进行分析，一起完成本次系统设计。

1.2 OV5640 初始化配置

首先是数据产生模块，也就是我们的 OV5640 摄像头。由于待显示的图像分辨率由 5 寸 TFT 屏的 800*480 变为了 VGA 的 1280*720，所以我们需要修改 OV5640 的初始化配置，使其输出相应分辨率的图像数据。OV5640 中控制输出图像分辨率大小的寄存器为 3808~380b，4 个寄存器功能如下：

表 1-1 输出大小窗口设置寄存器功能说明

寄存器地址	名称	默认值	功能描述
0x3808	TIMING DVPHO	0x00	bit[3:0]: 输出图像 X 方向尺寸大小的高 4 位
0x3809	TIMING DVPHO	0x10	bit[7:0]: 输出图像 X 方向尺寸大小的低 8 位
0x380a	TIMING DVPVO	0x00	bit[2:0]: 输出图像 Y 方向尺寸大小的高 3 位
0x380b	TIMING DVPVO	0x04	bit[7:0]: 输出图像 Y 方向尺寸大小的低 8 位

而在 OV5640 的初始化配置表中，这几个寄存器的值通过全局传参进行设

置：

```
module ov5640_init_table_rgb #(
    parameter DATA_WIDTH      = 24,
    parameter ADDR_WIDTH      = 8,
    parameter IMAGE_WIDTH      = 16'd640,
    parameter IMAGE_HEIGHT     = 16'd480,
    parameter IMAGE_FLIP_EN    = 1'b0,
    parameter IMAGE_MIRROR_EN  = 1'b0
)
    // DVPHO (<1280>500) (<640>280)IMAGE_WIDTH
    rom[223] = {16'h3808, IMAGE_WIDTH[15:8]};
    rom[224] = {16'h3809, IMAGE_WIDTH[ 7:0]}; // DVPHO
    // DVPVO (<720>2d0) (<480>1e0)IMAGE_HEIGHT
    rom[225] = {16'h380a, IMAGE_HEIGHT[15:8]};
    rom[226] = {16'h380b, IMAGE_HEIGHT[ 7:0]}; // DVPHO
```

因此我们需要修改设计顶层，将 IMAGE_WIDTH 和 IMAGE_HEIGHT 这两个参数的值设置为 720p 对应的参数：

```
parameter IMAGE_WIDTH  = 1280;
parameter IMAGE_HEIGHT = 720;
```

摄像头的输出帧率由寄存器 0x3035 控制，本次设计中我们不需要更改，也就是保持默认的 30 帧。

```
rom[208] = 24'h3035_21; // PLL 21:30fps 41:15fps 81:7.5fps
```

在摄像头被正确初始化后，会按照配置的参数，以 RGB565 格式，每秒输出 30 帧的 1280*720 分辨率的图像数据，同时输出的还有 pclk 和行场同步信号。数据会被输出给 DVP_Capture 模块，在 DVP_Capture 模块中数据被拼接为 16 位后通过 ddr3_ctrl_2port 模块写入 DDR3，并在需要时读出，随后送入到显示驱动模块。而 pclk 和场同步信号会送入到 camera_param_get 模块计算出 PCLK 频率与帧率，通过 VIO 显示。

1.3 摄像头帧率检测与显示

帧率检测使用的是 camera_param_get 模块，该模块的源码以及仿真文件用户可以在文档对应例程中提取。模块源码如下：

```
module camera_param_get(
    Clk,
    Rst_n,
    Pclk,
    VSync,

    Fps,
```

```
Fpclk
);

input Clk;
input Rst_n;
input Pclk;
input VSync;

output reg [7:0] Fps;
output reg [31:0] Fpclk;

parameter FCLK = 50_000_000;

reg [3:0] r_VSync;
always@(posedge Clk)
    r_VSync <= {r_VSync[2:0], VSync};

reg [31:0] sec_cnt;
//计时 1 秒
always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        sec_cnt <= 0;
    else if(sec_cnt >= FCLK - 1)
        sec_cnt <= 0;
    else
        sec_cnt <= sec_cnt + 1'd1;

//计数帧
reg [7:0] Fps_cnt;
always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        Fps_cnt <= 0;
    else if(sec_cnt == FCLK - 1)
        Fps_cnt <= 0;
    else if(r_VSync[3:2] == 2'b01)
        Fps_cnt <= Fps_cnt + 1'd1;
    else
        Fps_cnt <= Fps_cnt;

//输出帧统计结果
always@(posedge Clk)
    if(sec_cnt == FCLK - 1)
        Fps <= Fps_cnt ;
    else
        Fps <= Fps;

//考虑到 PCLK 有可能高于 FCLK，所以先预分频
```

```
reg[7:0]pre_div_cnt;
always@(posedge Pclk or negedge Rst_n)
if(!Rst_n)
    pre_div_cnt <= 0;
else
    pre_div_cnt <= pre_div_cnt + 1'd1;

reg[3:0]r_pre256_pclk;
always@(posedge Clk)
    r_pre256_pclk <= {r_pre256_pclk[2:0],pre_div_cnt[7]};

reg [23:0]pre_pclk_cnt;
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
    pre_pclk_cnt <= 0;
else if(sec_cnt == FCLK - 1)
    pre_pclk_cnt <= 0;
else if(r_pre256_pclk[3:2] == 2'b01)
    pre_pclk_cnt <= pre_pclk_cnt + 1'd1;
else
    pre_pclk_cnt <= pre_pclk_cnt;

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
    Fpclk <= 0;
else if(sec_cnt == FCLK - 1)
    Fpclk <= {pre_pclk_cnt,8'd0};
else
    Fpclk <= Fpclk;
endmodule
```

可以看到，代码通过在 1s 钟内计数 vsync 的个数来测量图像帧率；考虑到 PCLK 频率可能较大，对其 256 分频后计数其 1s 钟内的上升沿个数，然后将计数结果左移八位（乘以 256）得出 Fpclk 频率。

将 camera_param_get 模块添加进工程后，接下来添加一个 vio 核，用来显示 camera_param_get 模块输出的帧率信号 Fps 和 pclk 时钟频率 Fpclk。vio 核的配置如图 1-3 和图 1-4 所示：

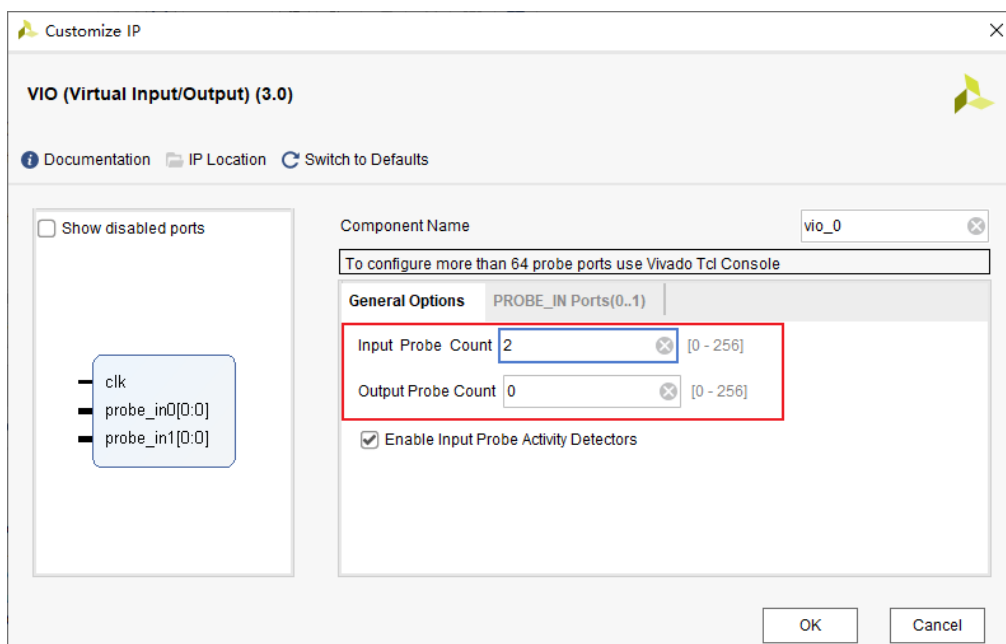


图 1-3 vio 常规配置

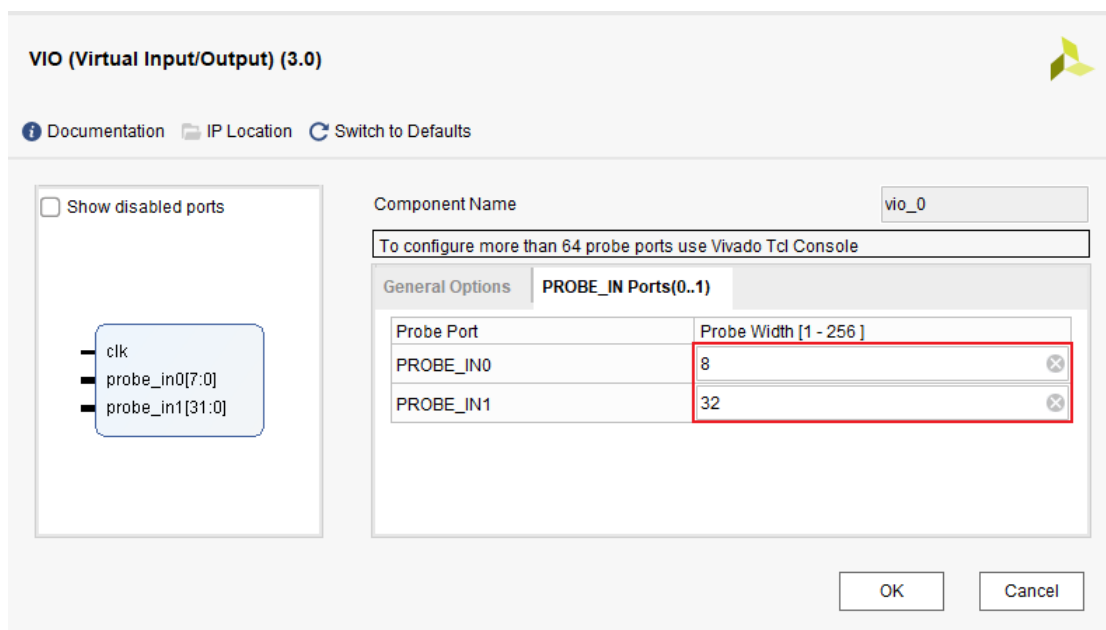


图 1-4 vio 输入引脚配置

probe_in0 用来连接 Fps 信号，probe_in1 用来连接 Fpclk 信号。配置完成后，将 camera_param_get 模块和 vio 核的例化添加到顶层中，相关代码如下：

```
//camera_param_get
wire [7:0]    Fps;
wire [31:0]   Fpclk;

camera_param_get camera_param_get(
    .Clk(loc_clk50m),
```



```
.Rst_n(~reset),
.Pclk(pclk_bufg_o),
.VSync(camera_vsync),

.Fps(Fps),
.Fpclk(Fpclk)
);

vio_0 vio_0(
    .clk(loc_clk50m),
    .probe_in0(Fps),
    .probe_in1(Fpclk)
);
```

在顶层中添加完例化后，我们也就完成了摄像头帧率检测与显示的设计，接下来就是产生数据显示所需的时序了，也就是配置显示驱动模块产生正确时序。

1.4 显示驱动模块配置

数据在被送到显示驱动模块后，显示驱动模块会跟据用户设置的参数，产生显示所需的时序信号。为了方便用户修改，显示驱动模块的配置参数全部以条件编译的方式保存于 disp_parameter_cfg.v 文件中。设计所需的显示分辨率为 1280*720，因此，我们需要通过行注释以及取消注释的方式，让配置表定义 1280*720 分辨率下的实现参数，注释以及取消注释后，部分代码如下：

```
//使用 4.3 寸 480*272 分辨率显示屏
//`define HW_TFT43
//使用 5 寸 800*480 分辨率显示屏
//`define HW_TFT50
//使用 VGA 显示器，默认为 640*480 分辨率，24 位模式，其他分辨率或需 16 位模式
//可在代码 63 行至 75 行进行重配置
`define HW_VGA
//=====
//以下宏定义选择用于根据显示设备进行位模式和分辨率 2 个参数的设置
//=====
`ifndef HW_TFT43 //使用 4.3 寸 480*272 分辨率显示屏
`define MODE_RGB565
`define Resolution_480x272 1 //时钟为 9MHz
`elsif HW_TFT50 //使用 5 寸 800*480 分辨率显示屏
`define MODE_RGB565
`define Resolution_800x480 1 //时钟为 33MHz
`elsif HW_VGA //使用 VGA 显示器，默认为 640*480 分辨率，24 位模式
//=====
//可选择其他分辨率和 16 位模式，需用户根据实际需求设置
```



```
//代码 70~71 行设置位模式
//代码 73~78 行设置分辨率
//=====
`define MODE_RGB888

// `define Resolution_640x480 1 //时钟为 25.175MHz
//`define Resolution_800x600 1 //时钟为 40MHz
//`define Resolution_1024x600 1 //时钟为 51MHz
//`define Resolution_1024x768 1 //时钟为 65MHz
`define Resolution_1280x720 1 //时钟为 74.25MHz
//`define Resolution_1920x1080 1 //时钟为 148.5MHz
`endif
```

可以看到这里通过条件编译将输出图像格式由 RGB565 设置为了 RGB888，将分辨率设置为了 1280*720。而根据条件编译的结果，在 191~204 行中，可以看到此时输出图像的时序参数如下：

```
191 `elsif Resolution_1280x720
192 `define H_Total_Time 12'd1650
193 `define H_Right_Border 12'd0
194 `define H_Front_Porch 12'd110
195 `define H_Sync_Time 12'd40
196 `define H_Back_Porch 12'd220
197 `define H_Left_Border 12'd0
198
199 `define V_Total_Time 12'd750
200 `define V_Bottom_Border 12'd0
201 `define V_Front_Porch 12'd5
202 `define V_Sync_Time 12'd5
203 `define V_Back_Porch 12'd20
204 `define V_Top_Border 12'd0
```

原设计中，disp_driver 模块的输入输出图像数据都是 RGB565 格式。而在我们修改显示驱动配置后，disp_driver 模块输入输出 RGB888 格式数据，因此我们需要修改顶层中输入输出图像数据信号的位宽以及对 disp_driver 模块例化的端口连接。同时为了方便区分，我们还需要将顶层中的所有名为 TFT 的信号修改为 VGA，由于 VGA 并不需要 TFT 的背光信号，因此，需要将顶层 TFT_pwm 信号相关的语句删除掉。

修改完成后，该部分代码如下：

```
output [23:0] VGA_rgb , //VGA 数据输出
output VGA_hs , //VGA 行同步信号
output VGA_vs , //VGA 场同步信号
output VGA_clk , //VGA 像素时钟
output VGA_de , //VGA 数据使能
```

```
wire [23:0] disp_data;
assign disp_data =
{rdfifo_dout[15:11], 3'd0, rdfifo_dout[10:5], 2'd0, rdfifo_dout[4:0], 3'd0}
;

disp_driver disp_driver
(
    .ClkDisp      (disp_clk      ),
    .Rst_p        (reset        ),
    .Data         (disp_data     ),
    .DataReq      (rdfifo_rden   ),
    .H_Addr       (              ),
    .V_Addr       (              ),
    .Disp_HS      (VGA_hs       ),
    .Disp_VS      (VGA_vs       ),
    .Disp_Red     (VGA_rgb[23:16] ),
    .Disp_Green   (VGA_rgb[15:8] ),
    .Disp_Blue    (VGA_rgb[7:0]  ),
    .Frame_Begin  (frame_begin  ),
    .Disp_DE      (VGA_de       ),
    .Disp_PCLK    (VGA_clk      )
);
```

1.5 产生 VGA 驱动时钟

修改完时序参数后的驱动模块并不能正常工作，因为此时该模块所使用的时钟仍是原来的 33MHz。根据前面我们得到的时序参数可以知道 1280*720 分辨率下扫描一行需要 1650 个时钟周期，一帧图像共有 750 行，进而我们能求出扫描一帧图像所需的时钟周期。而对于 VGA 显示器，其要求输入的图像信号刷新率不低于 50Hz，推荐采用 60Hz。也就是要求 1 秒钟内，显示屏上的图像更新 60 次。因此我们可以计算出满足该分辨率和帧率要求时，像素时钟频率为：

$$1650*750*60=74.25MHz$$

接下来使用设计中已有的 PLL 产生我们所需的 74.25MHz 时钟，代替原来的 33MHz 时钟（在原来的设计中该时钟分用于为 5 寸屏工作时钟），如图 1-5 所示：

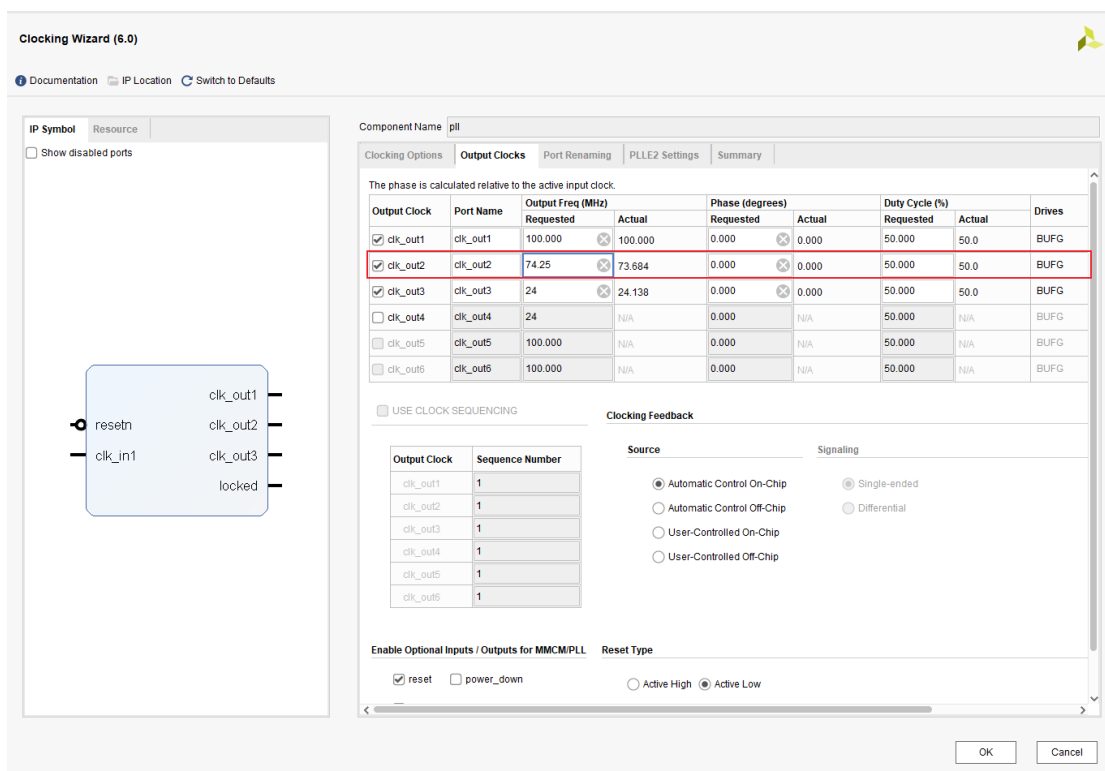


图 1-5 生成显示驱动时钟

至此，显示驱动模块的配置完成，模块会跟据设置好的参数，产生 1280*720@60Hz 分辨率图像显示所需的时序参数，将数据以及时序等信号输出给 VGA 输出模块。

1.6 修改顶层设计时钟

通过上述修改，我们已经完成了对 OV5640 以及显示驱动模块的相关配置。此时，对于设计来说，摄像头每秒钟会采集并输出 30 帧 1280*720 分辨率的图像数。这些数据会被存储进 DDR3 中，由显示驱动模块读取出来用于 VGA 显示。VGA 显示屏每秒钟会显示 60 帧分辨率为 1280*720 的图像（这里摄像头采集的每帧数据都会被读两次用于显示），据此我们可以计算出 DDR3 在一秒钟时间内所需的数据吞吐量为：

$$1280*720*30*16+1280*720*60*16 = 1327.104\text{Mbit}$$

在设计中，FPGA 的数据通过 fifo_axi4_adapter 模块，使用 AXI4 接口读写 PS 侧的 DDR3。原设计中，该模块突发传输时的数据位宽以及工作时钟如下：

```
fifo_axi4_adapter #(
    .FIFO_DW           (16),
    .WR_AXI_BYTE_ADDR_BEGIN (DDR_BASE_ADDR + 1'b1),
```

```
.WR_AXI_BYTE_ADDR_END    (DDR_BASE_ADDR +  
IMAGE_WIDTH*IMAGE_HEIGHT*2),  
.RD_AXI_BYTE_ADDR_BEGIN  (DDR_BASE_ADDR + 1'b1 ),  
.RD_AXI_BYTE_ADDR_END    (DDR_BASE_ADDR +  
IMAGE_WIDTH*IMAGE_HEIGHT*2),  
  
.AXI_DATA_WIDTH           (64                      ),  
.AXI_ADDR_WIDTH           (32                      ),  
.AXI_ID                   (4'b0000                ),  
.AXI_BURST_LEN            (8'd15                  ) //axi burst  
length = 16  
)fifo_axi4_adapter_inst  
(  
    //clock reset  
    .clk                    (loc_clk200m          ),  
    .reset                  (reset                 ),
```

可以看到突发读写的数据位宽为 64bit，工作时钟为 100MHz，因此我们可以粗略的计算出在绝对理想情况下，fifo_axi4_adapter 模块每秒钟所能实现的数据最大吞吐量为：

$$64 * 100M = 6400Mbit$$

对比可以发现，`fifo_axi4_adapter` 模块理论上的数据最大吞吐量近乎 5 倍于设计中 DDR 所需的吞吐量。但是实际上，数据从 PL 写入到 PS 还需要经历 AXI4 到 AXI3 协议的转换，且总线在传输时，本身也存在一定的时延。再加上 DDR 控制器本身在进行读、写、刷新等操作时，也会有一定的读写效率损坏。所以，实际上设计每秒所能达到的最大吞吐量要远小于 6400Mbit。在该时钟下实测，图像会存在由于带宽不足而导致的画面抖动情况，因此我们需要修改 `fifo_axi4_adapter` 模块的工作时钟为更大值，这里设置为 200MHz。

打开设计中的 PLL，将原来产生的 100MHz 时钟，修改为 200MHz，如图 1-6 所示：

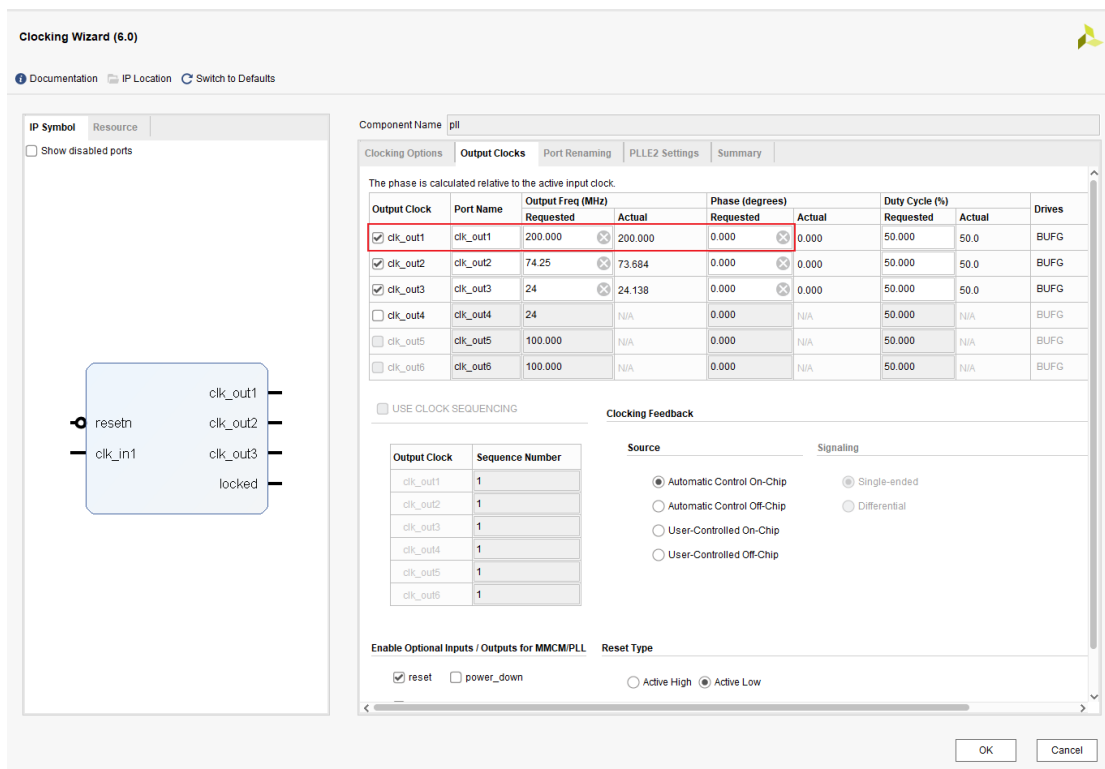


图 1-6 修改 PLL 时钟

修改完 PLL 时钟后，为了方便区分，需要将原设计中 100M 时钟信号 loc_clk100m 改为 loc_clk200m。该部分代码如下：

```

wire          loc_clk200m;
assign s_axi_aclk          = loc_clk200m;

//PS 先释放复位，PL 的逻辑复位释放往后延迟 20 个时钟周期
always@(posedge loc_clk200m or posedge reset_pre)
begin
if(reset_pre)
    reset_sync <= {20{1'b1}};
else
    reset_sync <= reset_sync << 1;
end

assign reset = reset_sync[19];

pll pll
(
    // Clock out ports
    .clk_out1 (loc_clk200m      ), // output clk_out1
    .clk_out2 (disp_clk        ), // 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
);

fifo_axi4_adapter #(
    .FIFO_DW              (16              ),
    .WR_AXI_BYTE_ADDR_BEGIN (DDR_BASE_ADDR + 1'b1 ),
    .WR_AXI_BYTE_ADDR_END   (DDR_BASE_ADDR +
IMAGE_WIDTH*IMAGE_HEIGHT*2),
    .RD_AXI_BYTE_ADDR_BEGIN (DDR_BASE_ADDR + 1'b1 ),
    .RD_AXI_BYTE_ADDR_END   (DDR_BASE_ADDR +
IMAGE_WIDTH*IMAGE_HEIGHT*2),

    .AXI_DATA_WIDTH        (64              ),
    .AXI_ADDR_WIDTH         (32              ),
    .AXI_ID                  (4'b0000        ),
    .AXI_BURST_LEN          (8'd15          ) //axi burst
length = 16
)fifo_axi4_adapter_inst
(
    //clock reset
    .clk              (loc_clk200m      ),
    .reset             (reset             ),
    //wr_fifo Interface
    .wrfifo_clr        (wrfifo_clr       ),
    .wrfifo_clk         (pclk_bufg_o     ),
    .....

```

完成对顶层设计修改后，要想设计能够正常工作，我们还需要为设计分配引脚，本次设计 VGA 输出模块的引脚分配表如表 1-2 所示：

表 1-2 VGA 输出模块管脚约束表

Pin Name	Signal Name	Pin NO.		Pin Name	Signal Name	Pin NO.
VGA_HS	VGA_hs	L15		VGA_G5	VGA_rgb[13]	G20
VGA_VS	VGA_vs	K14		VGA_G4	VGA_rgb[12]	G19
VGA_CLOCK	VGA_clk	K18		VGA_G3	VGA_rgb[11]	G17
VGA_BLK	VGA_de	H16		VGA_G2	VGA_rgb[10]	E19
VGA_R7	VGA_rgb[23]	E18		VGA_G1	VGA_rgb[9]	F17
VGA_R6	VGA_rgb[22]	D19		VGA_G0	VGA_rgb[8]	D20
VGA_R5	VGA_rgb[21]	E17		VGA_B7	VGA_rgb[7]	P19
VGA_R4	VGA_rgb[20]	D18		VGA_B6	VGA_rgb[6]	M18
VGA_R3	VGA_rgb[19]	H15		VGA_B5	VGA_rgb[5]	H17
VGA_R2	VGA_rgb[18]	F16		VGA_B4	VGA_rgb[4]	J18
VGA_R1	VGA_rgb[17]	J14		VGA_B3	VGA_rgb[3]	K19
VGA_R0	VGA_rgb[16]	G14		VGA_B2	VGA_rgb[2]	H18
VGA_G7	VGA_rgb[15]	H20		VGA_B1	VGA_rgb[1]	J19
VGA_G6	VGA_rgb[14]	G18		VGA_B0	VGA_rgb[0]	J20

分配完成后，VGA 输出模块就能正常进行工作了。数据在经由 VGA 线缆到达显示器的 VGA 接口后，对于模拟的 CRT 显示器，这些信号会直接被放大后用于驱动电子枪发射电子进而产生相应图像，而对于液晶显示器，则需要显示器使用专门的模拟数字转换芯片将模拟信号再转换为数字信号后，去驱动 RGB 接口的液晶显示屏显示图像。

至此整个设计梳理完成，接下来就可以生成比特流，将包含比特流的硬件资源描述文件导出到 SDK，随后烧录到开发板中进行验证了。

1.7 板级验证

本节介绍在 ACZ702 开发板上使用 OV5640 模块和 VGA 输出模块进行“OV5640 摄像头采集 VGA 显示屏显示”实验的验证。

1.7.1 所需硬件

1. ACZ702 开发板 x1
2. OV5640 摄像头 x1
3. VGA 输出模块 ACM_VGAHDMI x1
4. VGA 显示器 x1
5. 电源线 x1
6. VGA 线缆 x1

1.7.2 硬件连接

将 OV5640、VGA 输出模块、电源线依次连接开发板，将 VGA 缆线一端连接到 VGA 输出模块的 VGA 接口上，另一端连接 VGA 显示器。连接完成后将开发板电源开关拨到对应侧为开发板上电，如图 1-7 所示：

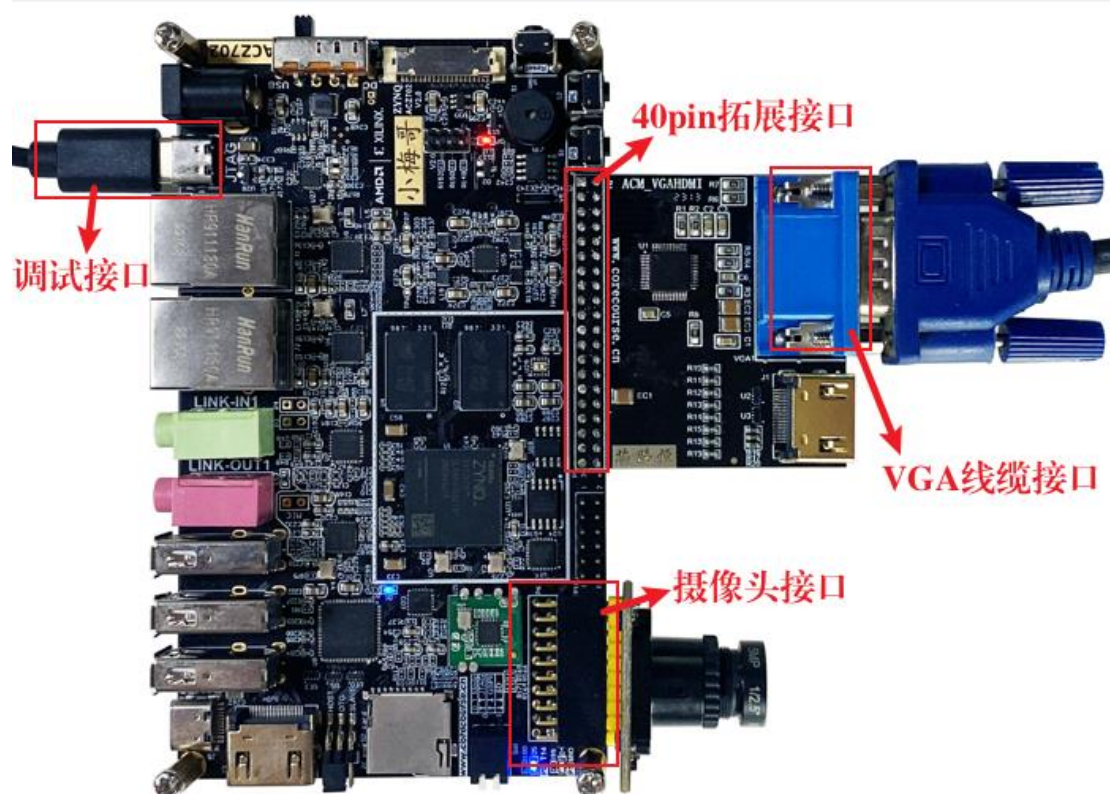


图 1-7 硬件连接图

在连接 VGA 接口时，需要注意：**VGA 线缆必须一端连接 GM7123 的 VGA 接口，另一端连接显示器或电视机的 VGA 接口。**

不可以将线缆一端连接 GM7123 的 VGA 接口，另一端连接到笔记本或者主机的 VGA 接口上。因为笔记本和主机的 VGA 接口为输出接口，而 GM7123 的 VGA 接口也为输出接口，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 VGA 接口为输入接口，GM7123 的 VGA 输出接口只有与这些输入接口相连，才能实现图像的显示。

1.7.3 效果展示

连接好硬件后，通过 SDK 将设计烧录到开发板中，稍等片刻后，显示效果如下图所示：

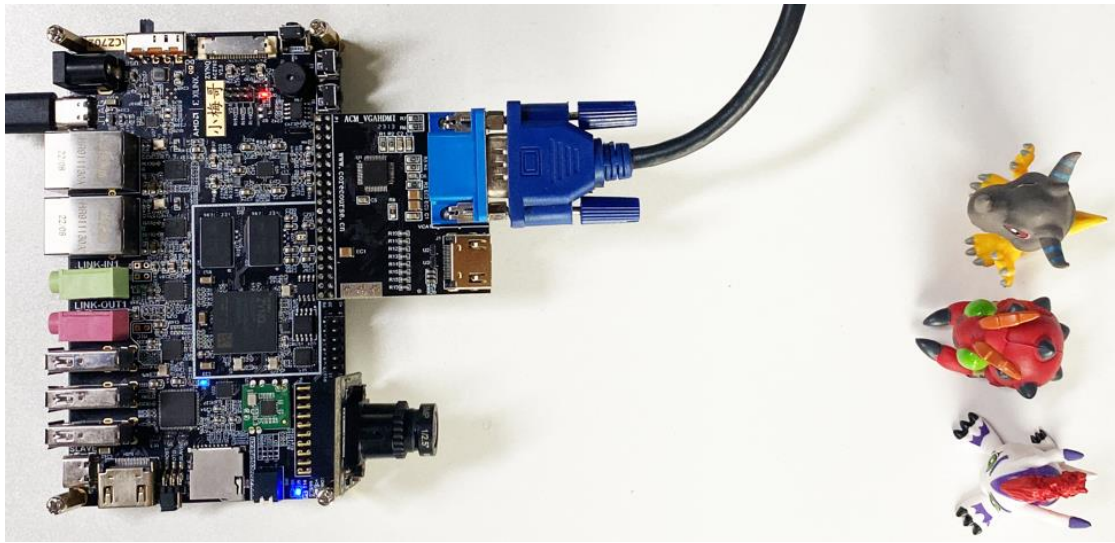


图 1-8 拍摄实物

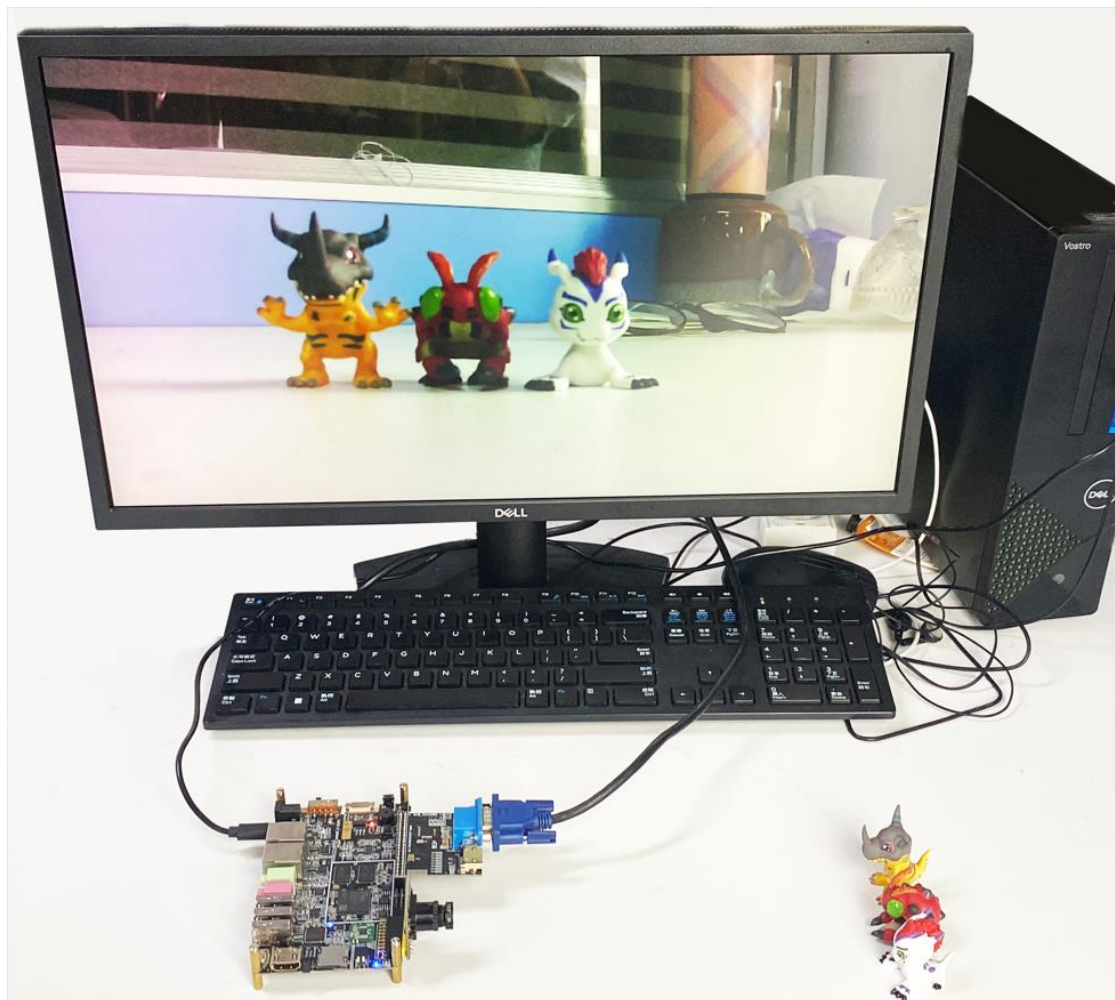


图 1-9 整体显示效果

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

下来打开 Vivado，连接硬件，在 VIO 窗口界面中添加摄像头帧率信号和像素时钟，测量的结果如图 1-10 所示：

hw_vios				
hw_vio_1				
Name	Value	Activity	Direction	VIO
> Fpclk[31:0]	[U] 84482816	↕	Input	hw_vio_1
> Fps[7:0]	[U] 30	↕	Input	hw_vio_1

图 1-10 摄像头帧率及输出像素时钟测量结果

可以看到，像素时钟约为 84.483MHz（该时钟与摄像头的开窗等寄存器有关，这里不带大家详细论证），图像帧率为 30 帧。这里，两个值虽然会上下浮动，但属于正常范围，因此本次设计成功。

这里简单解释一下两个值会上下浮动的缘由，PLL 为了能够同时尽可能满足各个时钟的需求，输出的实际时钟与我们所需时钟存在一定差异造成的，如图 1-6 所示。而解决的办法也非常简单，用户只需要保证最终输出给摄像头的是一个刚好 24MHz 的时钟即可。用户可以通过将 clocking wizard 的类型设置为 MMCM 以提高输出时钟的精确度，或是使用新的 PLL 单独为摄像头输出 24MHz 时钟。

1.8 总结

本章带大家实现了 OV5640 摄像头采集 VGA 显示屏显示系统的设计与验证，设计基于已有工程实现，虽然 TFT 屏的驱动原理与 VGA 驱动原理一致，但是使用时需要注意二者的接口位宽。