1 OV5640 摄像头采集 TFT 屏显示

章节导读

使用 OV5640，无外乎两个方面的工作——正确设置其工作模式和正确接收其输出的图像数据流。由于 CSI 接口需要专门的解码 IP 或者专用集成电路，所以这里不做讨论。在我们开发的 OV5640_V5 摄像头模组中，使用的是 DVP 接口。同时，为了让 OV5640_V5 模组能够正常的接受 FPGA 对其进行的各种工作模式设置，模组上也提供了 2 线制的 SCCB 接口（等价于 I2C）。所以对于 OV5640_V5 模块来说，只需要使用 I2C 控制器完成其各种模式寄存器设置，然后设计一个能够正确的接收 OV5640 的 DVP 接口数据流输出的数据的逻辑功能电路即可。

1.1 基于 OV5640 的图像采集显示系统

OV5640 具有 SCCB 接口和 DVP 接口时序。SCCB 接口实现很简单，由于兼容 I2C 总线的缘故，只需要使用成熟的 I2C 控制器配合简单的控制逻辑即可实现。而 DVP 接口的时序是很简单的，只需要 FPGA 被动的按照 DVP 接口的数据流输出时序把数据流渠道 FPGA 内部即可。

DVP 接口很简单，要想成功接收 DVP 接口的数据流非常容易，而关键点在于，接收到的数据流该去干什么。个人认为，脱离了实际应用场景来谈 DVP 接口数据流接收没有太大的意义。得出此结论是基于笔者使用 OV5640 开发的多个应用系统的经验。例如，如果需要把 OV5640 采集到的图像数据通过千兆以太网发送出去，由于千兆以太网本身就是 8 位的数据位宽，而 OV5640 的 DVP 接口也是 8 位位宽，所以几乎可以直接将 DVP 接口的数据线直连到千兆以太网发送逻辑端口上。而如果要将 OV5640 的图像数据通过 USB3.0 传输到 PC 机，由于 USB3.0 芯片与 FPGA 采用的是 32 位位宽的总线相连，那么将 DVP 接口的数据最好是能够接收到后每 4 个数据组合成一个 32 位的数据在送给 USB3.0 的接口逻辑。而最常见的将图像存储到 SDRAM 存储器中的应用，因为 SDRAM 提供的是 16 位的数据端口，所以将 DVP 接口输出的数据按照每 2 个数据组合成 16 位数据后再写入 SDRAM 比较合适。另外，从数据格式来说，输出的数据格式为 RGB888、RGB565、YUV422、YUV420、JPEG 等都有可能，不同的数据格式，在接收和使用时也有相应的要求。

为了让大家能够真正理解并掌握 OV5640 的应用开发规律。这里采用基于

实际应用的思路，先提出一个应用需求，然后根据应用需求来设计相应的接口逻辑。

1.2 图像采集显示系统定义

从实验现象的角度来说，将摄像头采集的图像直接显示在显示屏上是最简单直观的。所以本节将在实验“基于 DDR3 的串口传图帧缓存系统设计”实现一个图像采集显示应用系统，来设计 OV5640 应用系统中所需的基本应用逻辑电路。

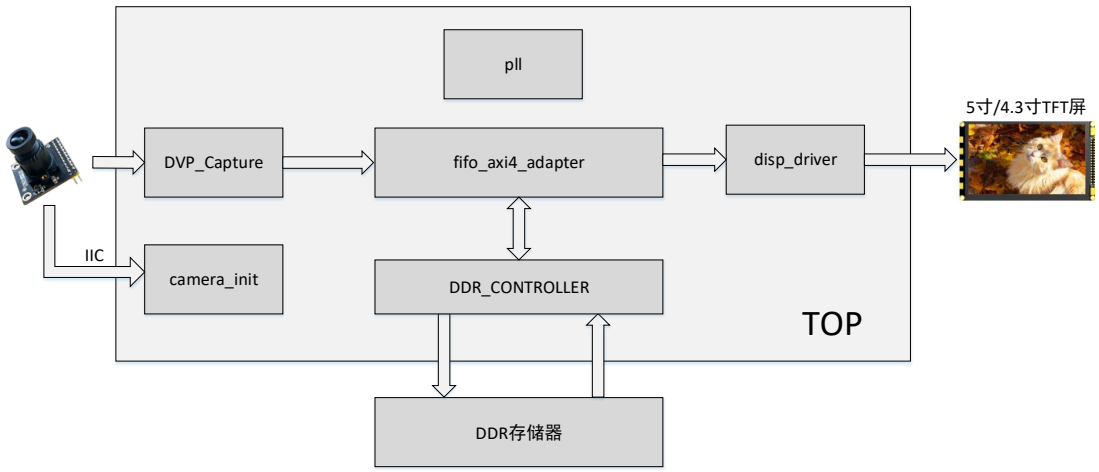


图 1-1 图像采集显示系统基本框图

上图为 OV5640 图像采集显示系统的框图，在该图中，所需的基本硬件只有 OV5640 摄像头，TFT 显示屏、AC201-SA5Z-50D0 开发板。对于这样的系统，分析理解时应按照数据流的流向进行。

按照数据流方向进行分析时，如果是为了理解现有系统的功能，可采用正推的方法进行，既按照数据流的流向顺序分析。而如果是为了根据系统模型来确定系统中的各个接口规范，则应该按照数据流的方向逆向分析，既从数据流的最终使用者一步一步倒推数据源的需求。此所谓由需求决定设计。

整个系统最终的目的是要在 TFT 显示屏上显示图像数据，由于 TFT 显示屏采用的是 RGB565 接口，为了让图像正常的显示在显示屏的指定位置，需要有相应的时序发生逻辑，这就是上图中的 Disp_Driver。该模块会按照 TFT 显示屏的接口时序产生对应的控制时序（HS、VS、DE、RGB data），实质上就是我们已经学习过的 TFT/VGA 控制器。

由于 TFT 显示屏显示时候，是按照 60 帧率的速率进行刷新，也就是每秒刷新 60 张图像，为了保证图像的正常显示，一般需要有一个图像缓冲区，TFT 控

制器实时从该图像缓冲区中读取数据并送往 TFT 屏上显示。所以在上述系统中，设计了一个 DDR3 用来存储需要显示的图像数据。

从设计的框图可以看出，整个系统与基于 DDR3 的串口传图帧缓存系统的差异在于串口传图是电脑通过串口将数据传输给 FPGA，而本实验是通过 OV5640 摄像头采集数据传给 FPGA 处理。本节重点是如何让驱动 OV5640，让 FPGA 正确采集到图像数据。

DVP Capture 模块负责将 DVP 接口输出的数据按照每两个一组合，得到符合 RGB565 图像格式的 16 位数据，之后数据处理过程与串口传图是一样，所以需要使用 SCCB 接口对 OV5640 进行设置，让其图像输出格式为 RGB565。

OV5640 要想能够正常的输出图像数据，必须经 SCCB 接口对其寄存器进行配置，所以整个系统首要的工作是使用控制逻辑经由 SCCB 接口对其进行初始化，在上图中，对 OV5640 进行初始化的功能模块为 Camera_Init。该模块会在系统开始工作时，对 OV5640 中各个寄存器写入指定值以实现初始化操作。接下来，就首先针对 Camera_Init 模块进行介绍。

1.3 摄像头初始化模块设计思路

所谓的 OV5640 摄像头初始化模块，其工作就是完成对 OV5640 中众多模式设置寄存器的写入操作。本质上就是使用 I2C 控制器将一些预先定义好的数据值写入到 OV5640 对应的寄存器中。所以我们完全可以使用之前设计好的 I2C 控制器加上一个存储好所有 OV5640 所需设置的寄存器参数的查找表，配合一定的控制逻辑实现，该设计实际比较简单，本节介绍时就不再采用思维引导的方式进行，而是介绍一个笔者设计好的功能模块的设计框架和应用方法。下图为我们设计的一个比较通用的 OV5640 初始化逻辑电路。

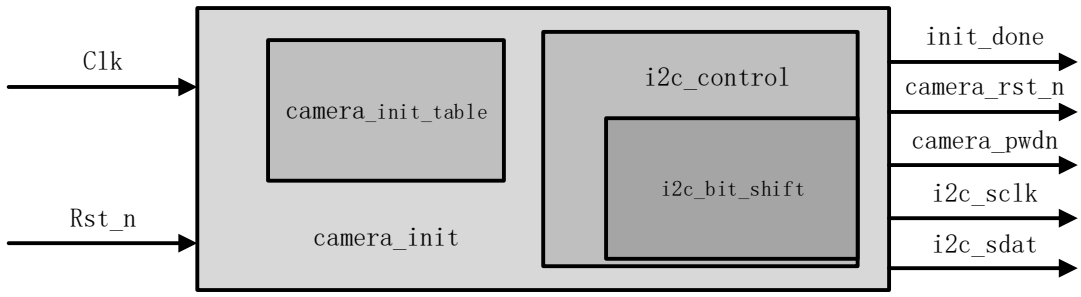


图 1-2 摄像头初始化模块系统框图

图中例化了设计好的通用 I2C 控制器（i2c_control）和一个 ROM 查找表（camera_init_table），在 camera_init 顶层逻辑中，通过简单的状态机循环读取

camera_init_table 中的数据值并通过 i2c_control 即可写入到 OC5640 的各个寄存器中。该代码已经提供在了我们的各个开发板配套资料包中，大家也可以通过我们的网站：www.corecourse.cn，搜索“camera_init”关键词找到。

1.4 摄像头初始化模块典型用法

在大多情况下，直接基于成熟的摄像头的初始化控制方案而稍加修改，就能将其应用到实战开发中。在应用上，仅需在整个系统顶层例化摄像头初始化模块的代码，即可正常使用该模块。比如，这里我们需要输出 160x128 的画幅，则顶层例化方法如下：

```
localparam RGB = 0;
localparam JPEG = 1;

parameter IMAGE_WIDTH  = 160; //图片宽度
parameter IMAGE_HEIGHT = 128; //图片高度(≤720)
parameter IMAGE_FLIP_EN   = 0;   //0：不翻转，1：上下翻转
parameter IMAGE_MIRROR_EN = 0;   //0：不镜像，1：左右镜像

//摄像头初始化配置
wire Init_Done;

camera_init
#(
    .CAMERA_TYPE ( "ov5640"    ),//"ov5640" or "ov7725"
    .IMAGE_TYPE  ( 0           ),// 0: RGB; 1: JPEG
    .IMAGE_WIDTH ( IMAGE_WIDTH  ),// 图片宽度
    .IMAGE_HEIGHT( IMAGE_HEIGHT ),// 图片高度
    .IMAGE_FLIP_EN ( 0           ),// 0: 不翻转，1: 上下翻转
    .IMAGE_MIRROR_EN  ( 0           ) // 0: 不镜像，1: 左右镜像
)camera_init
(
    .Clk         (Clk             ),
    .Rst_n       (locked          ),
    .Init_Done   (Init_Done       ),
    .camera_rst_n(                ),
    .camera_pwdn (                ),
    .i2c_sclk    (Camera_sclk     ),
    .i2c_sdat    (Camera_sdat     )
);
```

在代码中，加入了几个参数定义，每个参数对应了 OV5640 使用时的一个常用模式。现就这几个模式参数分别介绍如下。

CAMERA_TYPE	定义选定的摄像头是 ov5640 还是 ov7725。

RGB	定义 RGB 输出模式，是用来指示 IMAGE_TYPE 图片类型的，（0 表示为 RGB 类型，1 表示为 JPEG 类型），为了用户更直观，所以用了两个 localparam 来定义了，这样直接在 IMAGE_TYPE 的参数中直接填写 RGB 或者 JPEG，就能将 0 或者 1 传到被例化的模块里面。
JPEG	定义 JPEG 输出模式，是用来指示 IMAGE_TYPE 图片类型的，（0 表示为 RGB 类型，1 表示为 JPEG 类型），为了用户更直观，所以用了两个 localparam 来定义了，这样直接在 IMAGE_TYPE 的参数中直接填写 RGB 或者 JPEG，就能将 0 或者 1 传到被例化的模块里面。
IMAGE_WIDTH	输出图像宽度设置，当前寄存器初始化表中的设置，宽度最大可支持到 1280 像素
IMAGE_HEIGHT	输出图像高度设置，当前寄存器初始化表中的设置，宽度最大可支持到 720 像素
IMAGE_FLIP	图像翻转设置，0 则不翻转，1 则上下翻转
IMAGE_MIRROR	图像镜像设置，0 则不镜像，1 则左右镜像

使用时，关于控制图像的类型、分辨率、翻转、镜像，可以通过修改 localparam 和 parameter 的值，也可以直接在例化时候修改#后面括号里面的对应内容。我们上一个小节分析的摄像头配置信息，也是通过这里定义的参数，来传递给底层模块，从而实现对摄像头类型和配置信息的识别。

i2c_control 模块通过控制 i2c_bit_shift 模块顺序从 camera_init_table 里面（这里面有两个表 ov5640_init_table_jpeg、ov5640_init_table_rgb），读取预先存储好的摄像头的寄存器值，通过 i2c 总线（也就是顶层里面的 camera_sclk、camera_sdat 这两根线）来配置 ov5640 摄像头寄存器。

在接下来的内容中，我们将讲解几个典型的摄像头初始化 寄存器配置方法。如果读者希望了解更多初始化寄存器的相关内容，可以通过两个初始化寄存器的查找表文件 ov5640_init_table_jpeg 和 ov5640_init_table_rgb 中的寄存器的值更多信息，还可以参考 OV5640 的《OV5640_自动对焦照相模组应用指南(DVP_接口)__R2.13C.pdf》文档

1.5 摄像头初始化重点代码分析

1.5.1 摄像头初始化之寄存器控制

OV5640 支持多种输出格式，包括 JPEG、RGB、YUV、灰度、RAW 等。这些格式中，RGB、YUV、灰度、RAW 格式的寄存器初始化内容大体相同，仅有个别寄存器设置差异，可以使用同一个寄存器初始化列表，然后修改个别寄存器内容即可，而 JPEG 模式中，有很多寄存器的设置内容，相较于 RGB 格式差别比较大，所以我们设计了基于 generate 条件判断语法的选择性生成逻辑针对

选定的图像模式是 RGB 还是 JPEG，选择 RGB 或 JPEG 模式的寄存器初始化表，作为 OV5640 的寄存器初始化信息。

同时，由于 OV5640 和 OV7725 这一类 CMOS 摄像头的 DVP 接口都是一模一样的，差别仅在于 SCCB 接口以及寄存器初始化表，所以设计中同时还使用 generate 条件判断语法，对 OV7725 以及 OV5640 的初始化逻辑做了选择性生成。这样，我们在使用的时候，只需要在顶层定义好使用的摄像头型号以及图像模式，就能指导 FPGA 编译软件编译时只对于模式匹配的逻辑模块进行编译，该部分代码如下所示。

代码所在模块 ： camera_init

```
if(CAMERA_TYPE == "ov5640")
  begin
    assign device_id = 8'h78;
    assign addr_mode = 1'b1;
    assign addr = lut[23:8];
    assign wrdata = lut[7:0];

    if(IMAGE_TYPE == RGB) //------RGB------
    begin
      assign lut_size = 252;

      ov5640_init_table_rgb #(
        .IMAGE_WIDTH     (IMAGE_WIDTH     ),
        .IMAGE_HEIGHT    (IMAGE_HEIGHT    ),
        .IMAGE_FLIP_EN   (IMAGE_FLIP_EN   ),
        .IMAGE_MIRROR_EN (IMAGE_MIRROR_EN )
      )ov5640_init_table_rgb_inst
      (
        .addr (cnt ),
        .clk  (Clk ),
        .q    (lut )
      );
    end
    else  //------JPEG------
    begin
      assign lut_size = 250;

      ov5640_init_table_jpeg #(
        .IMAGE_WIDTH     (IMAGE_WIDTH     ),
        .IMAGE_HEIGHT    (IMAGE_HEIGHT    ),
        .IMAGE_FLIP_EN   (IMAGE_FLIP_EN   ),
        .IMAGE_MIRROR_EN (IMAGE_MIRROR_EN )
      )ov5640_init_table_jpeg_inst
      (
```

```
        .addr (cnt ),
        .clk  (Clk ),
        .q    (lut )
      );
    end
  end
  ////////////////////////////ov7725////////////////////////////////
  else if(CAMERA_TYPE == "ov7725")
  begin
    assign device_id = 8'h42;
    assign addr_mode = 1'b0;
    assign addr = lut[15:8];
    assign wrdata = lut[7:0];

    if(IMAGE_TYPE == RGB) //------RGB------
    begin
      assign lut_size  = 68;

      ov7725_init_table_rgb #(
        .IMAGE_WIDTH     (IMAGE_WIDTH     ),
        .IMAGE_HEIGHT    (IMAGE_HEIGHT    ),
        .IMAGE_FLIP_EN   (IMAGE_FLIP_EN   ),
        .IMAGE_MIRROR_EN (IMAGE_MIRROR_EN )
      )ov7725_init_table_rgb_inst
      (
        .addr (cnt ),
        .clk  (Clk ),
        .q    (lut )
      );
    end
  end
endgenerate
```

综合设计需求来看，IIC 控制器必须能够识别摄像头的类型为 ov5640，ov7725，也能够识别 OV5640 摄像头的数据格式为 RGB 或 jpeg。因此，如果要保证 IIC 控制器的兼容性，由于 OV5640 和 OV7725 的器件地址、地址长度、需要写入的寄存器个数都不一样，所以必须在 camera_init 层明确 IIC 控制器需要初始化的类型是 ov5640-RGB，ov5640-jpeg，ov7725-RGB 或 ov7725-jpeg。

如果摄像头的类型明确了，device_id,addr_mode,写入寄存器 rom 内的值的位宽、值的含义就明确了。进一步通过条件判断明确是 RGB 格式还是 jpeg 格式后，需填写的寄存器数量也明确了。

在上面的源码语句中，通过 generate——endgenerate 语句，实现了对摄像头类型、像素格式的选择。上述源码先判断了摄像头类型是 OV5640 还是 OV7725，

确认后，则对变量 device_id，addr_mode，addr，wrdata 四个变量进行赋值，以明确器件 ID，摄像头类型定义，摄像头寄存器地址和写入数据设定值。由于摄像头在 rgb 模式和 jpeg 模式下寄存器个数不同，所以必须先通过工程源码的顶层定义识别出是 RGB 模式还是 jpeg 模式，才能给出摄像头寄存器个数的准确定义数值。

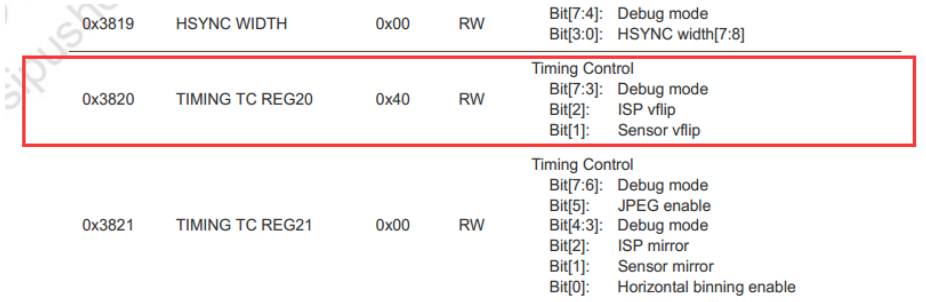
例如，顶层代码给出了摄像头的型号为 ov5640，通过查阅 ov5640 的技术手册，可以明确 ov5640 的 IIC 器件 ID 号为 device_id = 8’h78。此时定义 addr_mode = 1，addr = lut[23:8]，wrdata = lut[7:0]。addr_mode、addr 和 wrdata 共同配合，以便于给 ov5640 的寄存器地址赋初值。

代码所在模块 ： i2c_control
`assign reg_addr = addr_mode?addr:{addr[7:0],addr[15:8]};`

在 i2c_control 模块中，通过给变量 reg_addr 的赋值，就能够准确的告知 IIC 控制器向 ov5640 寄存器写入的数据格式，包括有效地址是低地址还是高地址在前。

由于 ov5640 摄像头配置寄存器位宽为 24 位，ov7725 摄像头配置寄存器位宽为 16 位，地址信息通过 addr_mode 和 addr 两个变量传递给 IIC 控制模块，即可在 IIC 模块中解析出摄像头类型和寄存器位宽。

又比如，以 ov5640_rgb 格式图像翻转控制器进行讲解。OV5640 摄像头数据手册定义寄存器地址号为 0x3820 控制成像是否翻转。官方手册指导：该寄存器默认值为 0x40，如果成像需要翻转，则将该寄存器默认值设为 0x47 或 0x46（表中未设定该寄存器最低位 Bit[0]的含义，所以该位为无关位）。



0x3819	HSYNC WIDTH	0x00	RW	Bit[7:4]: Debug mode Bit[3:0]: HSYNC width[7:8]
0x3820	TIMING TC REG20	0x40	RW	Timing Control Bit[7:3]: Debug mode Bit[2]: ISP vflip Bit[1]: Sensor vflip
0x3821	TIMING TC REG21	0x00	RW	Timing Control Bit[7:6]: Debug mode Bit[5]: JPEG enable Bit[4:3]: Debug mode Bit[2]: ISP mirror Bit[1]: Sensor mirror Bit[0]: Horizontal binning enable

图 1-3 OV5640 数据手册关于摄像头翻转控制的表格说明

代码所在模块 ： ov5640_init_table_rgb
`localparam IMAGE_FLIP_DAT  = IMAGE_FLIP_EN ? 8'h47 : 8'h40;`

从顶层传递而来的 IMAGE_FLIP_EN 信号，决定了写入的寄存器初始值是 8’h47 或 8’h40。

然后在摄像头初始化寄存器参数表中，以位拼接的格式，对 rom[211]寄存器进行赋值。lut 值的位宽决定了 rom 地址的位宽，一个 rom 地址的位宽为 24

位，高 16 位为地址号，低 8 位为初值。

理解了上面的内容，读者可以尝试结合代码，自行尝试修改 rom[211]，即 16’h3820 寄存器的值。读者可以分别尝试直接将 IMAGE_FLIP_DAT 替换为 8’h47，8’h46，8’h40，以直接观察输出效果获得图像翻转的直观体验。

1.5.1.1 JPEG 模式

```
rom[56] = 24'h4300_60;
...

rom[208] = 24'h4300_30; // YUV 422, YUYV
rom[209] = 24'h501f_00; // YUV 422

// 12824'h720, 30fps
// input clock 24Mhz, PCLK 42Mhz
rom[210] = 24'h3035_11; // PLL JPEG mode 11->15fps;21->7.5fps;
41->3.75fps;
rom[211] = 24'h3036_69; // PLL
rom[212] = 24'h3c07_07; // lightmeter 1 threshold[7:0]
rom[213] = {16'h3820, IMAGE_FLIP};    // flip
rom[214] = {20'h38212, IMAGE_MIRROR}; // no mirror
rom[215] = 24'h3814_11; // timing X inc
rom[216] = 24'h3815_11; // timing Y inc
rom[217] = 24'h3800_00; // HS
rom[218] = 24'h3801_00; // HS
rom[219] = 24'h3802_00; // VS
rom[220] = 24'h3803_00; // VS
rom[221] = 24'h3804_0a; // HW SET_OV5640 +  HE}
rom[222] = 24'h3805_3f; // HW SET_OV5640 +  HE}
rom[223] = 24'h3806_07; // VH SET_OV5640 +  VE}
rom[224] = 24'h3807_9f; // VH SET_OV5640 +  VE}
rom[225] = {16'h3808, IMAGE_WIDTH[15:8]}; // DVPHO  (<1280>500)
(<640>280)IMAGE_WIDTH
rom[226] = {16'h3809, IMAGE_WIDTH[ 7:0]}; // DVPHO
rom[227] = {16'h380a, IMAGE_HEIGHT[15:8]}; // DVPVO
(<720>2d0)  (<480>1e0)IMAGE_HEIGHT
rom[228] = {16'h380b, IMAGE_HEIGHT[ 7:0]}; // DVPHO
rom[229] = 24'h380c_0b; // HTS
rom[230] = 24'h380d_1c; // HTS
rom[231] = 24'h380e_07; // VTS
rom[232] = 24'h380f_b0; // VTS
rom[233] = 24'h3813_04; // timing V offset
rom[234] = 24'h3618_04;
rom[235] = 24'h3612_2b;
rom[236] = 24'h3709_12;
```

```
rom[237] = 24'h370c_00;
rom[238] = 24'h4004_06; // BLC line number
rom[239] = 24'h3002_00; // reset JFIFO, SFIFO, JPG
rom[240] = 24'h3006_ff; // disable clock of JPEG2x, JPEG
rom[241] = 24'h4713_03; // JPEG mode 3
rom[242] = 24'h4407_01; // Quantization scale
rom[243] = 24'h460b_35;
rom[244] = 24'h460c_22;
rom[245] = 24'h4837_16; // MIPI global timing
rom[246] = 24'h3824_02; // PCLK manual divider
rom[247] = 24'h5001_a3; // SDE on, CMX on, AWB on
rom[248] = 24'h3503_00; // AEC/AGC on
rom[249] = 24'h4740_20; // VS 1   //
```

1.5.1.2 RGB 模式

```
rom[56] = 24'h4300_61; // RGB565   //h4300_6f(千兆网模式)
...
// 12824'h720, 30fps
// input clock 24Mhz, PCLK 42Mhz
rom[208] = 24'h3035_21; // PLL  21:30fps  41:15fps  81:7.5fps
rom[209] = 24'h3036_69; // PLL
rom[210] = 24'h3c07_07; // lightmeter 1 threshold[7:0]
rom[211] = {16'h3820, IMAGE_FLIP}; // flip
rom[212] = {20'h38210, IMAGE_MIRROR}; // no mirror
rom[213] = 24'h3814_31; // timing X inc
rom[214] = 24'h3815_31; // timing Y inc
rom[215] = 24'h3800_00; // HS
rom[216] = 24'h3801_00; // HS
rom[217] = 24'h3802_00; // VS
rom[218] = 24'h3803_fa; // VS
rom[219] = 24'h3804_0a; // HW SET_OV5640 +  HE}
rom[220] = 24'h3805_3f; // HW SET_OV5640 +  HE}
rom[221] = 24'h3806_06; // VH SET_OV5640 +  VE}
rom[222] = 24'h3807_a9; // VH SET_OV5640 +  VE}
rom[223] = {16'h3808, IMAGE_WIDTH[15:8]}; // DVPHO  (<1280>500)
(<640>280)IMAGE_WIDTH
rom[224] = {16'h3809, IMAGE_WIDTH[ 7:0]}; // DVPHO
rom[225] = {16'h380a, IMAGE_HEIGHT[15:8]}; // DVPVO
(<720>2d0)  (<480>1e0)IMAGE_HEIGHT
rom[226] = {16'h380b, IMAGE_HEIGHT[ 7:0]}; // DVPHO
rom[227] = 24'h380c_07; // HTS
rom[228] = 24'h380d_64; // HTS
rom[229] = 24'h380e_02; // VTS
rom[230] = 24'h380f_e4; // VTS
rom[231] = 24'h3813_04; // timing V offset
rom[232] = 24'h3618_00;
```

```
rom[233] = 24'h3612_29;
rom[234] = 24'h3709_52;
rom[235] = 24'h370c_03;
rom[236] = 24'h3a02_02; // 60Hz max exposure
rom[237] = 24'h3a03_e0; // 60Hz max exposure
rom[238] = 24'h3a14_02; // 50Hz max exposure
rom[239] = 24'h3a15_e0; // 50Hz max exposure
rom[240] = 24'h4004_02; // BLC line number
rom[241] = 24'h3002_1c; // reset JFIFO, SFIFO, JPG
rom[242] = 24'h3006_c3; // disable clock of JPEG2x, JPEG
rom[243] = 24'h4713_03; // JPEG mode 3
rom[244] = 24'h4407_04; // Quantization scale
rom[245] = 24'h460b_37;
rom[246] = 24'h460c_20;
rom[247] = 24'h4837_16; // MIPI global timing
rom[248] = 24'h3824_04; // PCLK manual divider
rom[249] = 24'h5001_83; // SDE on, CMX on, AWB on
rom[250] = 24'h3503_00; // AEC/AGC on
rom[251] = 24'h4740_20; // VS 1
```

1.5.2 摄像头初始化之复位延时控制

OV5640 摄像头提供了专门的硬件复位管脚 camera_rst_n，通过拉低该管脚一定的时间，能够让 OV5640 的所有寄存器恢复到未经过配置的初始化状态。设计时，也推荐将该信号接到 FPGA 的管脚上，通过 FPGA 来控制。

但是在实际使用中发现，用户所使用的 FPGA 板不一样，有的摄像头接口能够给摄像头模块提供复位信号的控制，而有的则无对应管脚连接，所以无法对摄像头进行硬件复位。同时由于摄像头的供应商不一样，部分厂家的摄像头模块并未将这个硬件复位管脚引出，导致 FPGA 板即使有管脚，也无法控制到摄像头的硬件复位管脚。

为了解决这个问题，我们可以使用软件复位的方式。所谓软件复位，就是对摄像头的复位寄存器写入特定值，以设置摄像头进入复位状态。这样，无论 FPGA 和摄像头模式是否有条件进行复位，都能确保使用软件复位的方式，让摄像头进入正确的复位状态。

对于 OV5640 摄像头，其 0x3008 寄存器的 bit7 位就是软件复位位，只需要将该位置 1，然后等待一定时间（2~5ms）后再设置该位为 0，即可将 OV5640 的所有寄存器恢复到初始状态。

2.8 reset

The OV5640 sensor includes a **RESETB** pin that forces a complete hardware reset when it is pulled low (GND). The OV5640 clears all registers and resets them to their default values when a hardware reset occurs. A reset can also be initiated through the SCCB interface by setting register **0x3008**[7] to high.

Manually applying a hard reset upon power up is required even though on-chip reset is included. The hard reset is active low with an asynchronized design. The reset pulse width should be greater than or equal to 1 ms.

图 1-4 OV5640 摄像头技术手册对复位方式的官方描述

table 7-1　system and IO pad control registers (sheet 3 of 7)

address	register name	default value	R/W	description
0x3007	CLOCK ENABLE03	0xFF	RW	Clock Enable Control (0: disable clock; 1: enable clock) Bit[7]: Enable digital gain compensation clock Bit[6]: Enable SYNC FIFO clock Bit[5]: Enable ISPFC SCLK clock Bit[4]: Enable MIPI PCLK clock Bit[3]: Enable MIPI clock Bit[2]: Enable DVP PCLK clock Bit[1]: Enable VFIFO PCLK clock Bit[0]: Enable VFIFO SCLK clock
0x3008	SYSTEM CTROL0	0x02	RW	System Control Bit[7]: Software reset Bit[6]: Software power down Bit[5:0]: Debug mode
0x3009	DEBUG MODE	–	–	Debug Mode

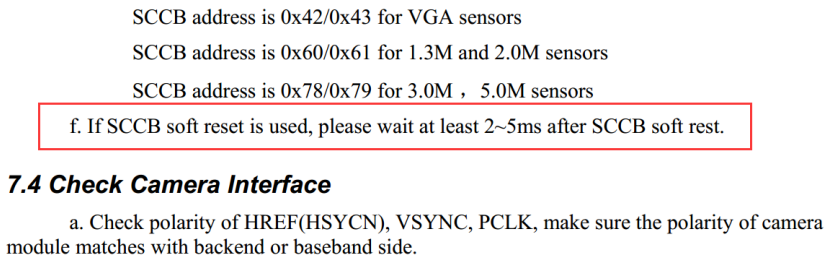
图 1-5 OV5640 摄像头数据手册对于复位寄存器 0x3008 的赋值描述

具体来说，OV5640 摄像头提供了硬件复位管脚复位和软件写复位寄存器来复位两种复位方式。

硬件复位：通过对 OV5640 芯片的复位管脚拉低 1ms 左右实现完成复位

软件复位：通过 0x3008 寄存器的 bit[7]位写入 1 来进入复位，写入 0 来退出复位。

从官方的技术手册要求来看，采用软件复位而放弃对 OV5640 复位管脚的控制，并不是无条件的。官方手册要求：OV5640 如果进行软件复位，需在配置复位寄存器后，产生至少 2ms 至 5ms 的等待延时。下方即为官方手册对该要求的原文：



SCCB address is 0x42/0x43 for VGA sensors

SCCB address is 0x60/0x61 for 1.3M and 2.0M sensors

SCCB address is 0x78/0x79 for 3.0M ， 5.0M sensors

f. If SCCB soft reset is used, please wait at least 2~5ms after SCCB soft rest.

7.4 Check Camera Interface

a. Check polarity of HREF(HSYCN), VSYNC, PCLK, make sure the polarity of camera module matches with backend or baseband side.

图 1-6 OV5640 Camera Module Hardware Application Notes 关于复位延时的描述原文

合理的复位延时，能保证 rom[1]后续寄存器值的正确写入，否则，如果设

置软件复位后立即退出复位，继续写后续寄存器，会因为当前摄像头还处在复位逻辑中，继而导致此期间写寄存器的值不会生效。这样就会导致最后紧跟在复位操作后的若干个寄存器没有正确初始化，从而影响正确的成像。我们的初始化表的第二个寄存器，就是对复位寄存器进行设置，以让 OV5640 进入复位状态。

代码所在模块 ： ov5640_init_table_rgb

```
rom[1 ] = 24'h3008_82; // software reset, bit[7]
//delay 5ms
```

综合以上分析，可知上方代码中 rom[1]的意义，以及正确实现软件复位需要的 5ms 延时要求。

如果能够通过程序正确进行上述操作，那么 OV5640 的硬件复位管脚 camera_rst_n 即使不连接 FPGA 管脚，而是以硬件的方式设置为永久的高电平，也可以正常工作。

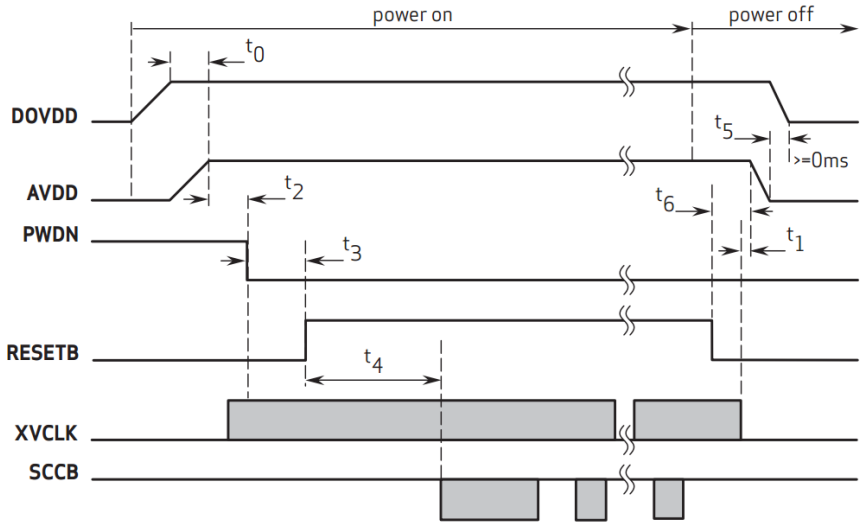


图 1-7 ov5640 摄像头稳定采集数据时 PWDN 和 RESETB 为固定的初始值即可

接下来，我们解决如何在模块内部给出 camera_pwdn 和 camera_rst_n 的初值定值，同时通过设定延时时长的方法，保证复位控制寄存器 rom[1]设定后能有至少 5ms 的确认时间的问题。

代码所在模块 ： camera_init

```
assign camera_pwdn = 0;
```

代码所在模块 ： camera_init

```
assign camera_rst_n = 1;
```

给定初值后，即需开始解决写入 rom[1]后延时 5ms 的问题。首先，我们看

到 i2c_control 模块中，设计的状态机有控制数据写入完成后延时的功能。

代码所在模块 ： i2c_control

```
WAIT_DLY:
begin
    if(dly_cnt <= dly_cnt_max) begin
        dly_cnt <= dly_cnt + 1'b1;
        state <= WAIT_DLY;
    end
    else begin
        dly_cnt <= 0;
        RW_Done <= 1'b1;
        state <= IDLE;
    end
end
```

在 i2c_control 模块中，我们设置了等待延迟状态，并通过 dly_cnt 计数器来完成时钟周期的计数等待。当 dly_cnt 计数器完成延时，即计数到和 dly_cnt_max 相等时，i2c 状态机才会回到写 i2c 操作的起始状态进而开始下一次的数据读写。这样看来，我们可以通过设定 dly_cnt_max 的值，来控制延迟计数器的计数次数，从而设定一次 i2c 读写完成后的等待延迟时间。

从 ov5640 的寄存器配置手册来看，大部分寄存器配置都没有延迟要求。或者一次 i2c 的读写和存储，并不会影响到其他后续寄存器的写入效果，而唯独复位寄存器较为特殊。而 rom 地址，在 camera_init 模块中，又是和变量 cnt 对应的。这样，在 camera_init 模块的设计中，我们就可以将 rom 地址和延迟总时长对应起来，实现专门对 rom[1]进行一个较长时间的延迟。

代码所在模块 ： camera_init

```
always@(posedge Clk or negedge Rst_n)
  if(~Rst_n)
    cnt <= 0;
  else if(Go)
    cnt <= 0;
  else if(cnt < lut_size)begin
    if(RW_Done && (!ack))
      cnt <= cnt + 1'b1;
    else
      cnt <= cnt;
  end
  else
    cnt <= 0;

......     ......
  reg [1:0]state;
```

```
  always@(posedge Clk or negedge Rst_n)
  if(~Rst_n)begin
    state <= 0;
    wrreg_req <= 1'b0;
    i2c_dly_cnt_max <= 32'd0;
  end
  else if(cnt < lut_size)begin
    case(state)
      0:
        if(Go)
          state <= 1;
        else
          state <= 0;
      1:
        begin
          wrreg_req <= 1'b1;
          state <= 2;
          if(cnt == 1)
//////给出满足延时不小于 5ms 的 16 进制计数器值//////
            i2c_dly_cnt_max <= 32'h40000;
          else
            i2c_dly_cnt_max <= 32'd0;
        end
      2:
        begin
          wrreg_req <= 1'b0;
          if(RW_Done)
            state <= 1;
          else
            state <= 2;
        end
      default:state <= 0;
    endcase
  end
  else
        state <= 0;
```

我们在进行 rom[1]的寄存器赋值操作时，即当 cnt 为 1 时，专门对于 cnt 延迟控制计数器进行特殊赋值，即对 i2c_dly_cnt_max 赋值为 32’h40000，而对其他 rom 地址保持 i2c_dly_cnt_max 为默认值 0。由于主时钟的时钟周期为 20ns，这样，计数 32’h40000,就是计数十进制的 262144 拍，就可以满足大于 5ms 的延迟。

通过例化过程，camera_init 模块的 i2c_dly_cnt_max，和 i2c_control 模块的 dly_cnt_max 信号对应，完成计数器最大值设置的传递。

代码所在模块 ： camera_init

```
i2c_control i2c_control(
    .Clk         (Clk             ),
    .Rst_n       (Rst_n           ),
    ......  ......
    .dly_cnt_max (i2c_dly_cnt_max ),
    .i2c_sclk    (i2c_sclk        ),
    .i2c_sdat    (i2c_sdat        )
);
```

这样，i2c_control 模块就可以直接通过已有的延时状态，来实现 rom[1]寄存器写完后产生不低于 5ms 的延时这一任务。随之就不会发生 rom[1]复位配置过程中，rom[2]，rom[3].....又在同时写入的现象。

1.6 硬件管脚复位和 Init_done 信号的生成

上一小节我们讲解 ov5640 摄像头的寄存器控制复位实现方法。在本节中，我们将讲解 ov5640 硬件复位控制方法，并通过硬件复位结束的延时信号，将 Init_done 信号强制拉低，直到所有摄像头初始化寄存器写入完成后，再拉高。

AC201-SA5Z-50D0 开发板在设计之初，也是带有摄像头专用复位管脚引出的。FPGA 给出硬件复位信号，摄像头会立马进行复位。因为只有硬件复位完成后，寄存器配置操作才有可能开始，所以硬件管脚复位会提前于寄存器软件复位操作而到来。

OV5640 官方技术手册要求，摄像头模组需上电复位（注意，这里的复位和寄存器配置上的复位不同，是指硬件管脚复位）完成 20ms 后再配置摄像头。原始文档的时序图如下：

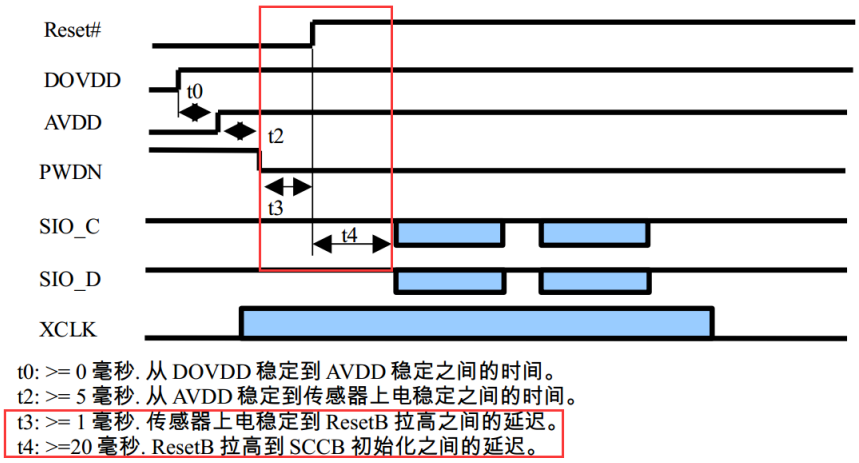


图 1-8 ov5640_自动对焦照相模组应用指南对上电、复位和 SCCB 初始化的延时要求

在实际硬件复位设计时，只需要使用一个计数器作延时使用，即可实现上述时序图中的延时要求。以下为 camera_init 代码中设计的复位时序的相关代码。

针对以上要求，我们作出如下设计：

代码所在模块 ： camera_init

```
//上电并复位完成 20ms 后再配置摄像头，所以从上电到开始配置应该是 1.0034 + 20
= 21.0034ms
//这里为了优化逻辑，简化比较器逻辑，直接使延迟比较值为 24'h100800，是
21.0125ms
always@(posedge Clk or negedge Rst_n)
if(~Rst_n)
    delay_cnt <= 21'd0;
else if (delay_cnt == 21'h100800)
    delay_cnt <= 21'h100800;
else
    delay_cnt <= delay_cnt + 1'd1;

//当延时时间到，开始使能初始化模块对 OV5640 的寄存器进行写入
assign Go = (delay_cnt == 21'h1007ff) ? 1'b1 : 1'b0;
```

上方代码注释很清晰的告诉了我们设定延时最大值的来历。每个时钟周期，delay_cnt 自加 1，直到 delay_cnt 值达到 21'h100800，如果 delay_cnt == 21'h1007ff，说明在下一个时钟周期，delay_cnt == 21'h100800 条件即将满足，将 Go 拉高一个时钟周期。

代码所在模块 ： camera_init

```
always@(posedge Clk or negedge Rst_n)
if(~Rst_n)
    Init_Done <= 0;
else if(Go)
    Init_Done <= 0;
else if(cnt == lut_size)
    Init_Done <= 1;
```

Init_Done 信号值的判断，优先判断 Go 的值，Init_Done 初始值为 0，在下一时钟周期，delay_cnt == 21'h100800 条件得到满足，而 Go 立即从 1 变为 0，开始等待 rom 寄存器配置完成。当满足条件 cnt == lut_size 表明所有寄存器配置完成，Init_Done 信号拉高输出。

1.7 OV5640 DVP 接口时序逻辑设计

上一节，我们已经完成了 OV5640 初始化逻辑的介绍。接下来，将要开始完成 DVP 接口的时序设计。

为了便于分析 DVP 接口。这里以 OV5640 输出 3*3 像素的图像大小矩阵为

例绘制时序图，介绍 DVP 接口的时序，并分析接口逻辑设计方法。

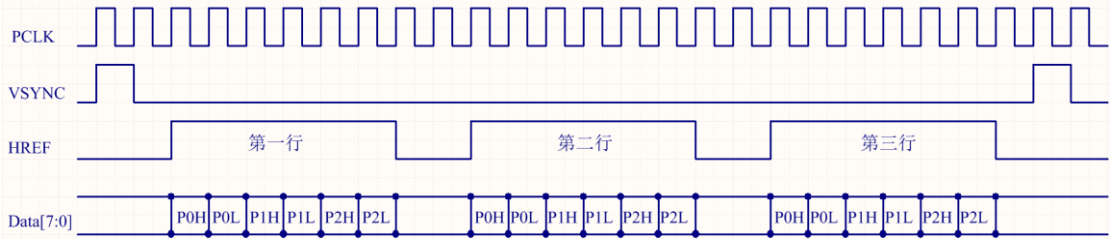


图 1-9 OV5640DVP 接口时序逻辑图

特别说明：上图仅为时序示意图，其中 VSYNC 的高电平脉冲宽度，VSYNC 下降沿到 HREF 上升沿之间的间隔时钟个数，HREF 下降沿到下一个 HREF 的上升沿之间的高电平之间的时钟个数，以及最后一个 HREF 下降沿到 VSYNC 上升沿之间的间隔时钟个数实际上远大于图中所绘制，绘制时为了保证画幅大小和进行了精简，这些差异不影响我们分析 DVP 接口时序。

- VSYNC 的高脉冲标志着新一帧图像数据的即将到来。所以当 VSYNC 高脉冲出现之后，第一个 HREF 高电平期间 DATA 端口上传输的就是整幅图像的第一行数据，紧接着第二行，直到最后一行输出完成，再产生 VSYNC 的高脉冲开始新一帧图像数据的输出。
- HREF 上升沿后的第一个时钟时刻的数据为该行图像的第一个像素数据的高字节（P0H），第二个时钟时刻的数据为第一个像素数据的低字节（P0L），以此类推，直到第 2N 个数据为第 N 个像素数据的低字节，然后一行数据输出完成，HREF 变为低电平。间隔一定时钟周期后再开始下一行图像数据的输出。
- Data 数据线上，PxH 和 PxL 两个 8 位的数据拼接为一个 RGB565 像素的图像数据。

1.7.1 基本数据流接收

根据应用需求，整个 DVP Capture 模块的设计目的就是要实现每两个数据拼接为 1 个 16 位的数据并按照写 RAM 或 FIFO 的接口形式输出，模块设计框图如下图所示：

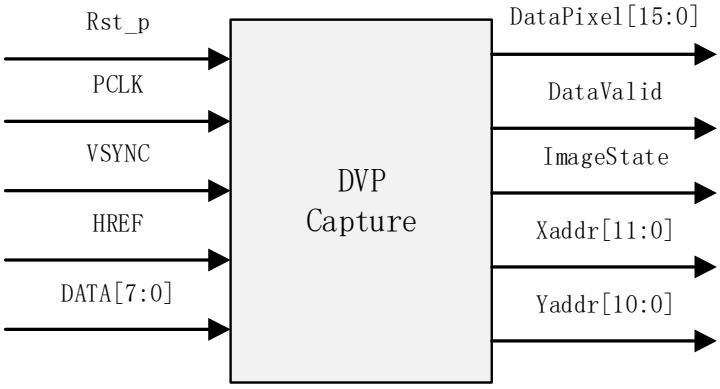


图 1-10 DVP 接口数据流接收模块输入输出接口

模块中，DataPixel 为 16 位的 RGB565 格式的像素数据，由连续的 2 个 Data 数据拼接而来。DataValid 为 DataPixel 数据有效标志信号，由于 DATA 端口需要 2 个时钟才能传输一个像素所需的 16 位数据，所以 DataPixel 端口上的数据理论来说应该是每 2 个时钟周期只有一个时钟周期是真正有效的，所以 DataValid 在连续的两个时钟周期中，只有一个时钟周期为高电平，一个时钟周期为低电平，来确保下一级在使用 DataPixel 时，每两个时钟周期内只使用一次，使用时，如果是数据直接写入 FIFO 或者 RAM，可以直接将 DataValid 信号当做 wrreq 信号使用。下图为该模块的时序图：

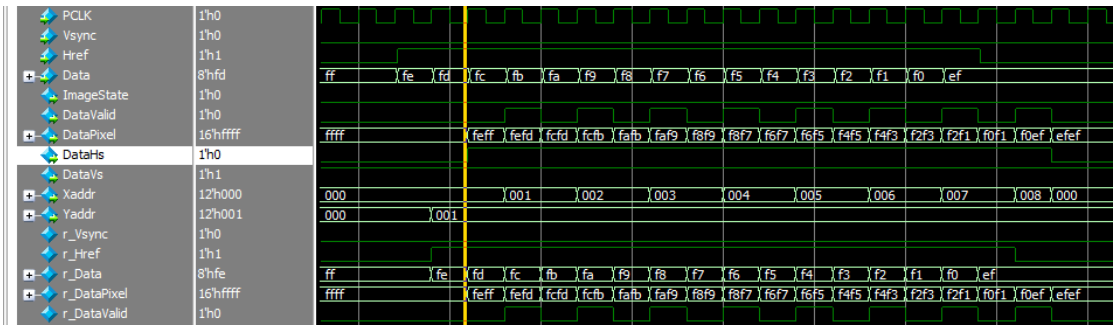


图 1-11 DVP 接口模块时序图

从图中可以看到，DataPixel 的数据在 DataValid 信号为高电平的时候，输出的是前两个时钟周期的值拼接成的。如 C1 和 C2 组成一个 16 位的数据，在 C2 之后延迟两个数据输出在 DataPixel 端口上。这样就完成了 DVP 接口数据流转 RGB565 数据流的功能。

1.7.2 像素位置输出

在有些应用中，需要实时知道当前输出的图像数据在输出像素矩阵中的绝对位置，以便于根据不同的位置进行不同的处理。最典型的如使用图像处理技

术定位图像中某个点的具体位置。则需要用到该信息，因此，该 DVP Capture 模块也将每个像素对应的位置通过 Xaddr 和 Yaddr 两个端口输出。在上图中以 C1C2 这个数据为例，可以看到，在 DataValid 为高电平的时候，数据位 C1C2，而此时的 Xaddr 也为 1。每当 DataValid 高电平信号到来时，Xaddr 的值加 1。当一行图像数据输出完成之后，Xaddr 清零。

Xaddr 和 Yaddr 的生成非常简单。对于 Xaddr 来说，只要设计一个 Hcount 计数器，在 HREF 为高电平期间持续计数即可，然后舍去将 Hcount 的最低位得到的值就刚好可以与像素输出时刻一一对应。而对于 Yaddr，则只需要使用一个 Vcount 计数器对 HREF 的上升沿进行计数即可。关于该部分实现的具体细节详见提供的设计代码的完整内容。

需要关注的是，在模块中，Xaddr 和 Yaddr 都是从 1 开始算的，并非从 0 开始，也就是说，输出的每行最开头的一个像素对应的 Xaddr 的值为 1，输出的每帧图像的第一行数据，其 Yaddr 的值也为 1。

1.7.3 舍弃前 N 张图像

在某些工程师的应用说明中有提到，一般 CMOS 摄像头开始工作后，其起始的几张图像是不稳定的。关于这种不稳定的情况，有两种解读，一是输出的图像数据量不稳定，例如本来应该每行输出 800 个像素点的数据，但是实际上只输出了不到 800 个或者超过 800 个数据。另一种解读是刚开始输出的这几幅图像数据量稳定，但其颜色失真较大，所以不建议取用。在笔者实际设计调试的过程中，即使直接使用起始时候的图像，也暂未发现输出数据量异常的情况，图像数据量应该是稳定的。另外对于颜色失真较大的可能，由于没有专门采集和分析起始几张图像的质量，也暂未得到认证。为了兼顾这种可能出现的情况。在 DVP Capture 中还是对起始的 10 帧图像做了舍弃处理。使用一个 FrameCnt 计数器计数 VSYNC 的上升沿，每个上升沿就意味着一帧图像即将开始，当该计数值达到 10 之后，在使能 DataValid 信号，从而确保了前 10 帧图像的数据不会输出。关于该部分实现的具体细节详见提供的设计代码的完整内容。

1.7.4 系统异常状态恢复控制

在实际应用的过程中，笔者还遇到了这样一个问题。那是在将图像数据写入 fifo 的一个应用中，由于系统运行过程中突然被复位，然后重新开始运行，此时 fifo 中已经有写入了一部分数据还没来得及读走，如果 fifo 没有在系统复位

后也执行相应的清零操作，这部分数据会依旧保留在 fifo 内，读取端在复位之后，默认会认为 fifo 中的第一个数据就是 Xaddr 和 Yaddr 都为 1 的那个数据，如果读取端不借助 Xaddr 和 Yaddr 的值，就会导致真正的第一个数据的位置判断错误。为了简化逻辑设计并解决这个问题，可以通过合理的设计，让系统从复位中恢复之后，该 FIFO 在摄像头的第一个数据（这里指摄像头开始工作后 DVP Capture 模块输出的一个数据，非每帧或每行图像的第一个数据）开始向其写入之前，先执行一次清零操作，让 FIFO 中的残留数据先清零，然后再向其中写入时，就能与 FIFO 的读取端对位置的判定方式一致了。所以在逻辑中设计了一个图像状态信号 ImageState，当系统进入复位状态后该信号变为高电平，只有当系统从复位中恢复且 DVP 接口的第一个 VSYNC 高电平出现时，再将该信号拉低，实际在应用过程中，可以直接将 ImageState 信号连接到 FIFO 的清零端口（aclr），就能实现对 FIFO 的合理清零了，关于该部分实现的具体细节详见提供的设计代码的完整内容。

整个设计逻辑的代码如下所示：

```
module DVP_Capture(
  Rst_p,
  PCLK,
  Vsync,
  Href,
  Data,

  ImageState,
  DataValid,
  DataPixel,
  DataHs,
  DataVs,
  Xaddr,
  Yaddr
);
  input Rst_p;
  input PCLK;
  input Vsync;
  input Href;
  input [7:0] Data;

  output reg ImageState;
  output DataValid;
  output [15:0] DataPixel;
  output DataHs;
  output DataVs;
  output [11:0] Xaddr;
```

```
output [11:0] Yaddr;

reg       r_Vsync;
reg       r_Href;
reg [7:0] r_Data;

reg [15:0]r_DataPixel;
reg       r_DataValid;
reg       r_DataHs;
reg       r_DataVs;
reg [12:0]Hcount;
reg [11:0]Vcount;
reg [3:0] FrameCnt;

reg       dump_frame;

//等到初始化摄像完成且头场同步信号出现，释放清零信号，开始写入数据
always@(posedge PCLK or posedge Rst_p)
if (Rst_p)
  ImageState <= 1'b1;
else if(r_Vsync)
  ImageState <= 1'b0;

//对 DVP 接口的数据使用寄存器打一拍，以用信号边沿检测功能
always@(posedge PCLK)
begin
  r_Vsync <= Vsync;
  r_Href  <= Href;
  r_Data  <= Data;
end

//在 HREF 为高电平时，计数输出数据个数
always@(posedge PCLK or posedge Rst_p)
if(Rst_p)
  Hcount <= 0;
else if(r_Href)
  Hcount <= Hcount + 1'd1;
else
  Hcount <= 0;

/*根据计数器的计数值奇数和偶数的区别，在计数器为偶数时，
将 DVP 接口数据端口上的数据存到输出像素数据的高字节，在计
数器为奇数时，将 DVP 接口数据端口上的数据存到输出像素数据
的低字节*/
always@(posedge PCLK or posedge Rst_p)
if(Rst_p)
  r_DataPixel <= 0;
```

```
  else if(!Hcount[0])
    r_DataPixel[15:8] <= r_Data;
  else
    r_DataPixel[7:0] <= r_Data;

  /*在行计数器计数值为奇数，且 HREF 高电平期间，产生输出
   数据有效信号*/
  always@(posedge PCLK or posedge Rst_p)
  if(Rst_p)
    r_DataValid <= 0;
  else if(Hcount[0] && r_Href)
    r_DataValid <= 1;
  else
    r_DataValid <= 0;

  always@(posedge PCLK)
  begin
    r_DataHs <= r_Href;
    r_DataVs <= ~r_Vsync;
  end

  /*使用 Vcount 计数器对 HREF 信号的高电平进行计数，统计
  一帧图像中的每一行图像的行号*/
  always@(posedge PCLK or posedge Rst_p)
  if(Rst_p)
    Vcount <= 0;
  else if(r_Vsync)
    Vcount <= 0;
  else if({r_Href,Href} == 2'b01)
    Vcount <= Vcount + 1'd1;
  else
    Vcount <= Vcount;

  /*输出 X 地址*/
  assign Yaddr = Vcount;

  /*由于一行 N 个像素的图像输出 2N 个数据，所以 Hcount 计数
  值为 N 的 2 倍，将该计数值除以 2 后即可作为 Xaddr 输出*/
  assign Xaddr = Hcount[12:1];

  /*帧计数器，对每次系统开始运行后的前 10 帧图像进行计数*/
  always@(posedge PCLK or posedge Rst_p)
  if(Rst_p)
    FrameCnt <= 0;
  else if({r_Vsync,Vsync}== 2'b01)begin
    if(FrameCnt >= 10)
      FrameCnt <= 4'd10;
```

```
    else
      FrameCnt <= FrameCnt + 1'd1;
  end
  else
    FrameCnt <= FrameCnt;

  /*舍弃每次系统开始运行后的前 10 帧图像的数据，以确保输出图像稳定*/
  always@(posedge PCLK or posedge Rst_p)
  if(Rst_p)
    dump_frame <= 0;
  else if(FrameCnt >= 10)
    dump_frame <= 1'd1;
  else
    dump_frame <= 0;

  assign DataPixel = r_DataPixel;
  assign DataValid = r_DataValid & dump_frame;
  assign DataHs = r_DataHs;
  assign DataVs = r_DataVs;

endmodule
```

同时，设计一个 DVP 信号发生测试逻辑，用来对设计的 DVP 接口逻辑进行仿真测试，该部分设计代码思路很简单，只要按照 DVP 接口的时序要求产生 PCLK、HREF、VSYNC、DATA 端口的数据即可，该测试文件代码如下所示：

```
`timescale 1ns/1ns

module DVP_Capture_tb;

  reg         Rst_p;
  reg         PCLK;
  reg         Vsync;
  reg         Href;
  reg  [7:0] Data;

  wire        ImageState;
  wire        DataValid;
  wire [15:0]DataPixel;
  wire        DataHs;
  wire        DataVs;
  wire [11:0]Xaddr;
  wire [11:0]Yaddr;

  DVP_Capture DVP_Capture(
    .Rst_p      (Rst_p      ),//input
    .PCLK       (PCLK       ),//input
    .Vsync      (Vsync      ),//input
```

```
    .Href       (Href       ),//input
    .Data       (Data       ),//input     [7:0]

    .ImageState (ImageState ),//output reg
    .DataValid  (DataValid  ),//output
    .DataPixel  (DataPixel  ),//output    [15:0]
    .DataHs     (DataHs     ),//output
    .DataVs     (DataVs     ),//output
    .Xaddr      (Xaddr      ),//output    [11:0]
    .Yaddr      (Yaddr      ) //output    [11:0]
);

initial PCLK = 1;
always#40 PCLK = ~PCLK;

parameter WIDTH = 16;
parameter HIGHT = 12;

integer i,j;

initial begin
  Rst_p = 1;
  Vsync = 0;
  Href  = 0;
  Data  = 8'hff;
  #805;
  Rst_p = 0;
  #400;

  repeat(15)begin
    Vsync = 1;
    #320;
    Vsync = 0;
    #800;
    for(i=0;i<HIGHT;i=i+1)
    begin
      for(j=0;j<WIDTH;j=j+1)
      begin
        Href = 1;
        Data = Data - 1;
        #80;
      end
      Href = 0;
      #800;
    end
  end
  $stop;
```

```
  end
endmodule
```

1.8 系统板级测试

在顶层设计分析综合没有错误并且顶层仿真确认设计功能没有问题后，进行上板验证。对工程的管脚和时钟进行约束后，生成 Bit 文件。上板调试硬件平台基于 AC201-SA5Z-50D0 开发板。AC201-SA5Z-50D0 开发板连接示意图如下：

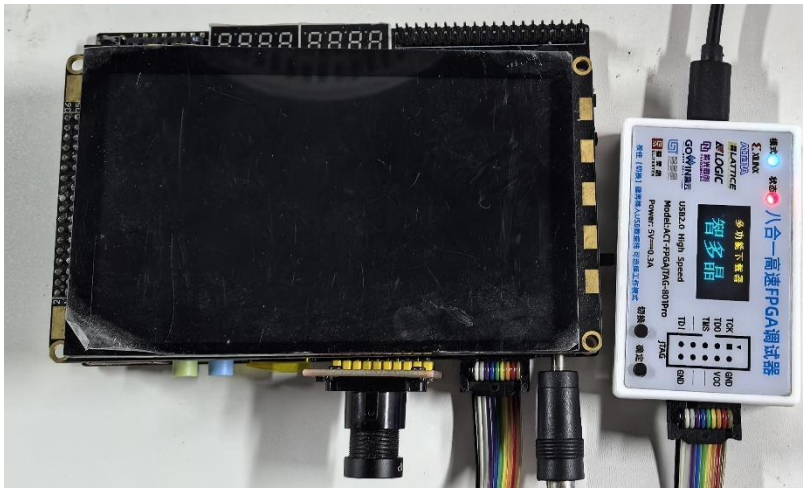


图 1-12 板级验证开发板连接图

连接好开发板后，下载生成的 Bit 文件。下载完成后， TFT 屏上会显示摄像头采集的视频图像。

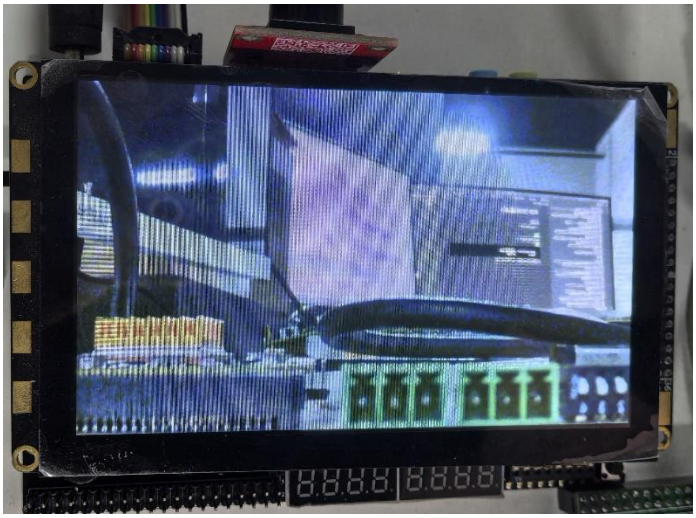


图 1-13 板级验证实验效果