# 双目OV5640摄像头RGB TFT-LCD显示实验

在双目OV5640 摄像头VGA显示实验中，我们成功地在VGA显示器上实时显示出了双目摄像头采集的图像。本章我们将使用FPGA开发板实现对双目OV5640摄像头双路的数字图像采集，并通过液晶屏模块实时显示。本章包括以下几个部分：

* 1. 简介
  2. 实验任务
  3. 硬件设计
  4. 程序设计
  5. 下载验证

## 简介

前面我们做过“双目OV5640摄像头VGA显示实验”，对双目摄像头的概念做了简介，如果大家还不够熟悉，请回去查看。本次实验我们将在前面“双目OV5640摄像头VGA显示”的基础上学习双目OV5640摄像头LCD显示。

## 实验任务

本章我们利用双目OV5640摄像头采集图像，将采集到的图像实时显示在RGB LCD屏幕上，两幅图像分别占据LCD显示屏的左右半边，兼容正点原子提供的五种屏幕：4.3寸480\*272，4.3寸800\*480，7寸800\*480，7寸1024\*600，10.1寸1280\*800，同时我们需要在屏幕上叠加相应的字符。

## 硬件设计

我们的开拓者开发板上有两个扩展口，分别是P6和P7，它们都可以用来连接双目OV5640摄像头。本次实验我们只能选P7扩展口，不能选用P6扩展口，因为我们的LCD模块的部分管脚是和P6口共用的。P7扩展口原理图如图 1.3.1所示：可以看出该扩展口一共48个管脚，而本次实验我们将使用其中的40个，包括电源。

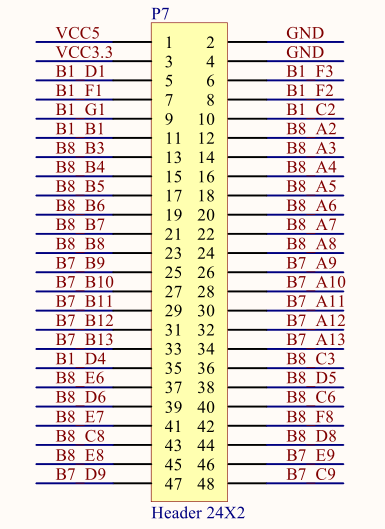


图 1.3.1 P7扩展口原理图

ATK-Dual-OV5640是正点原子推出的一款双目OV5640摄像头模块，该模块通过2\*20排母（2.54mm间距）同外部连接，连接时将双目摄像头的排母直接插在开发板上的P7扩展口即可，模块实物如图 1.3.2所示：

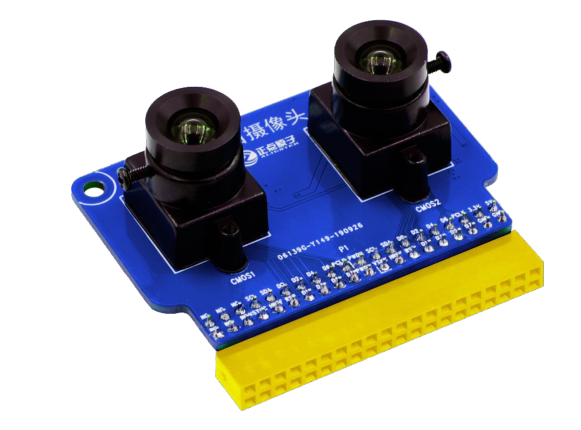


图 1.3.2双目摄像头模块实物图

以下是我们给出的双目摄像头模块的管脚分配表，方便大家对照查看。

表1.3.1双目摄像头管脚分配

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 方向 | 管脚 | 端口说明 |
| cam0\_pclk | input | B7 | cmos0数据像素时钟 |
| cam0\_vsync | input | A13 | cmos0场同步信号 |
| cam0\_href | input | A12 | cmos0行同步信号 |
| cam0\_data[0] | input | B11 | cmos0数据 |
| cam0\_data[1] | input | A10 | cmos0数据 |
| cam0\_data[2] | input | B10 | cmos0数据 |
| cam0\_data[3] | input | A9 | cmos0数据 |
| cam0\_data[4] | input | B9 | cmos0数据 |
| cam0\_data[5] | input | A8 | cmos0数据 |
| cam0\_data[6] | input | B8 | cmos0数据 |
| cam0\_data[7] | input | A7 | cmos0数据 |
| cam0\_rst\_n | output | A11 | cmos0复位信号，低电  平有效 |
| cam0\_pwdn | output | A6 | cmos0电源休眠模式选  择信号 |
| cam0\_scl | output | B13 | cmos0 SCCB\_SCL线 |
| cam0\_sda | inout | B12 | cmos0 SCCB\_ SDA线 |
| cam1\_pclk | input | D1 | cmos1数据像素时钟 |
| cam1\_vsync | input | A5 | cmos1场同步信号 |
| cam1\_href | input | A4 | cmos1行同步信号 |
| cam1\_data[0] | input | B3 | cmos1数据 |
| cam1\_data[1] | input | A2 | cmos1数据 |
| cam1\_data[2] | input | B1 | cmos1数据 |
| cam1\_data[3] | input | C2 | cmos1数据 |
| cam1\_data[4] | input | G1 | cmos1数据 |
| cam1\_data[5] | input | F2 | cmos1数据 |
| cam1\_data[6] | input | F1 | cmos1数据 |
| cam1\_data[7] | input | F3 | cmos1数据 |
| cam1\_rst\_n | output | A3 | cmos1复位信号，低电  平有效 |
| cam1\_pwdn | output | B6 | cmos1电源休眠模式选  择信号 |
| cam1\_scl | output | B5 | cmos1 SCCB\_SCL线 |
| cam1\_sda | inout | B4 | cmos1 SCCB\_ SDA线 |

## 程序设计

根据实验任务，首先我们设计如图 1.4.1所示的系统框图，本章实验的系统框架延续了“双目OV5640摄像头VGA显示实验”的整体架构。系统包括以下模块：PLL时钟模块，SDRAM控制器模块，两个摄像头驱动模块和LCD驱动模块以及图像尺寸模块。其中PLL时钟模块，摄像头驱动模块以及SDRAM控制器模块较“双目OV5640摄像头VGA显示”没有做修改，本次实验做的修改位于LCD驱动模块以及图像尺寸模块。



图 1.4.1 系统架构

下面分别介绍各个模块。

PLL时钟模块（pll\_clk）：PLL时钟模块通过调用锁相环（PLL）IP核实现，总共输出3个时钟，频率分别为100Mhz、100Mhz（SDRAM相位偏移时钟）和100Mhz（LCD分频用）时钟。100Mhz时钟和100Mhz相位偏移时钟作为SDRAM读写控制模块的驱动时钟，最后一个100Mhz时钟作为LCD

顶层模块的驱动时钟。

摄像头驱动模块（ov5640\_dri）：本模块由IIC驱动，OV5640 IIC配置，摄像头图像采集封装而成，与“双目OV5640摄像头显示实验”相同，如图 1.4.2所示。



图 . 摄像头驱动模块

由于本次实验连接了两个相同的摄像头，因此我们会对摄像头驱动模块例化两次。需要注意的是：由于本次实验需要兼容五个RGB LCD屏幕的分辨率，我们对OV5640 IIC配置模块的相关参数不再固定为一定值，而是由不同的RGB LCD屏幕决定，参数具体由单独一个模块产生。

LCD顶层模块（lcd）：LCD顶层模块负责驱动LCD显示器正确显示。它由四个子模块封装而成。如图 1.4.3所示。



图 . LCD模块

从图中我们可以看出这四个模块分别是读ID模块，LCD显示模块，LCD驱动模块以及时钟分频模块。各模块的功能简单介绍如下：

读ID模块（rd\_id）：识别并读出不同RGB LCD屏幕的ID。

LCD显示模块（lcd\_disply）：产生RGB LCD要显示的像素内容，包括图像内容和叠加上去的字符。

LCD驱动模块（lcd\_driver）：产生驱动RGB LCD显示的各种驱动信号。

钟分频模块（clk\_div）：针对不同的RGB LCD屏幕产生不同的驱动时钟。

LCD顶层模块将是我们本次实验主要介绍的内容。在这里我们注意一下，上图各模块之间的箭头方向并不完全反映各模块之间真正的数据交互情形，只是示意。

图像尺寸模块（picture\_size）：该模块负责根据不同的LCD屏幕传进来的不同ID产生不同的摄像头图像配置参数，包括图像横向分辨率，纵向分辨率，图片大小，以及HTS（行长），VTS（帧长）。我们会在后面详细介绍。

SDRAM控制器模块（sdram\_top）：它由两个子模块组成，分别是sdram\_controller模块和sdram\_fifo\_ctrl模块。sdram\_controller模块：该模块实现了复杂的SDRAM读写操作，并向外留出了必要的用户接口，是用户和SDRAM内部进行数据交互的桥梁，本模块的内容与之前模块完全相同，这里不做介绍。

sdram\_fifo\_ctrl模块：这是一个重要模块，它负责从摄像头采集到FIFO输出的数据流的控制。我们在“双目OV56摄像头VGA显示实验”中对这一模块有详细介绍。本次实验我们将不做介绍。

以上就是我们对整个系统框图的及各模块的介绍。在具体介绍程序设计之前，我们来直观了解一下工程代码文件组织形，模块之间的调用关系。我们在Quartus II下打开本工程，如图 1.4.4，

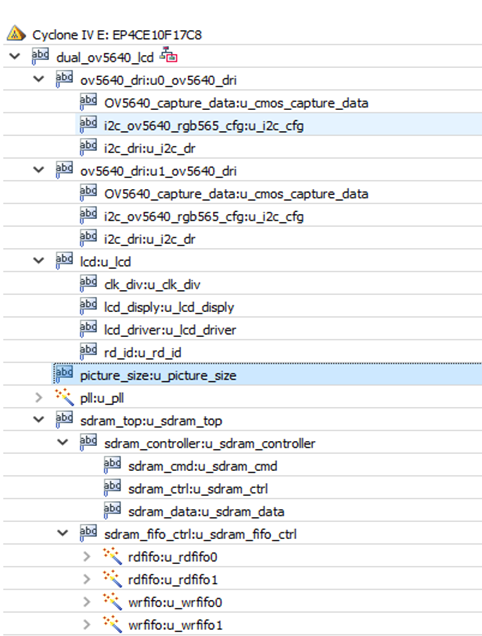


图 1.4.4 OV5640双目摄像头LCD显示工程目录

最后如果我们在工程下执行Tool→Netlist→Viewer RTL Viewer，就会看到整个工程代码综合出来的RTL图，如图 1.4.5所示。结合它我们可以更细一步了解端口连接，模各块划分以及数据流向。由于RTL图较大，我们无法在文档中显示足够清晰的图片，大家可以在Quartus II下自行查看。

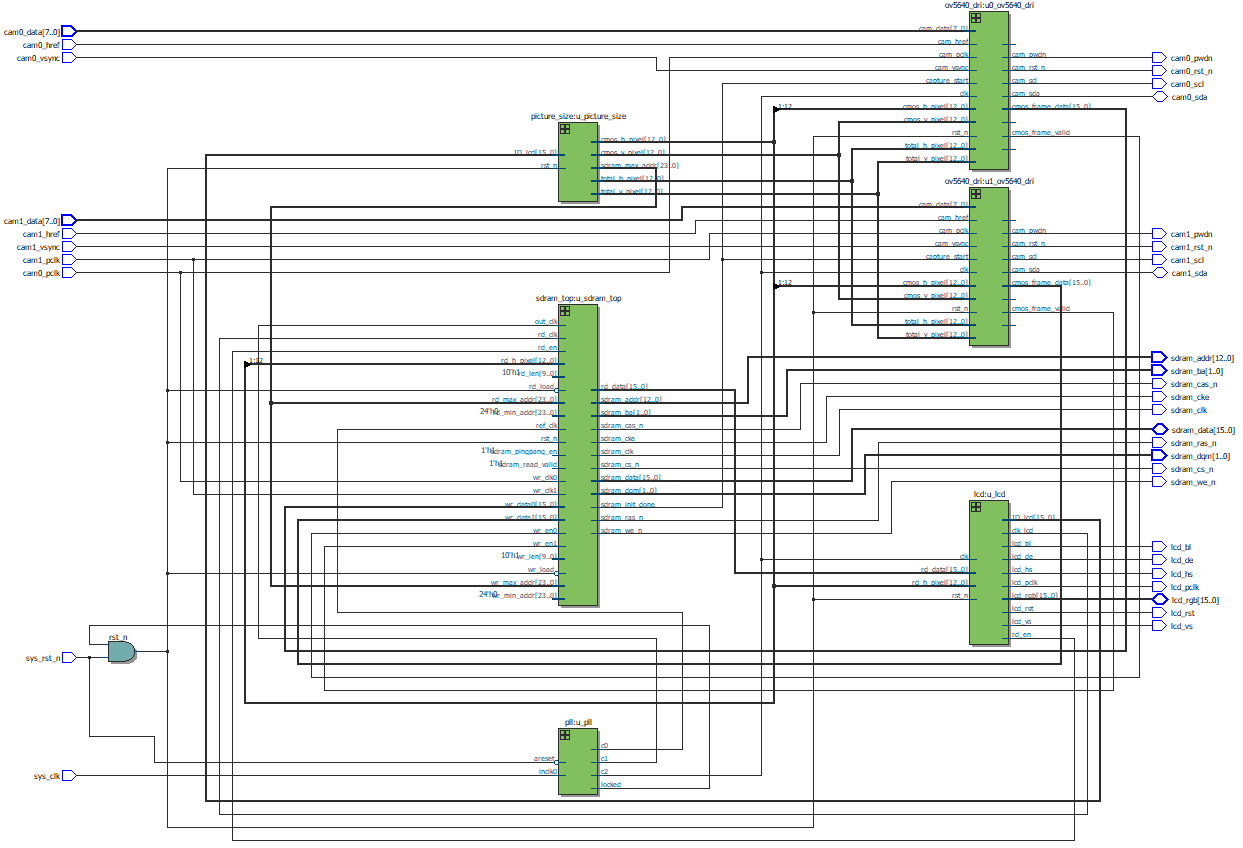


图 1.4.5 OV5640双目摄像头LCD显示RTL图

本次实验是在“双目OV5640摄像头VGA显示实验”的基础上作的修改。从摄像头的图像采集到SDRAM的缓存，FIFO的读写控制，我们完全沿用了上一次实验的代码，不同点是设备的显示及驱动部分。上一次实验我们用VGA显示器显示，显示器是固定的一种分辨率，而本次实验我们采用RGB LCD作为显示器，面对三个新的问题：第一，RGB LCD的驱动和VGA驱动略有不同，第二，要兼容五种不同的RGB LCD屏幕，第三，在图像上叠加字符。基于这些问题，我们设计了图像尺寸模块（picture\_size）和LCD顶层模块（lcd），本实验我们将主要介绍这两个模块。首先来讲解图像尺寸模块（picture\_size）。

我们在开拓者“OV5640摄像头RGB TFT-LCD显示实验”中使用过图像尺寸模块（picture\_size），并做了详细讲解。本次实验我们在原有模块的基础上做的一处改动，下面我们只贴出改动部分的代码。

26 ID\_4342 **:** **begin**

27 cmos\_h\_pixel **=** 13'd480**;**

28 cmos\_v\_pixel **=** 13'd256**;**

29 sdram\_max\_addr **=** 23'd122880**;**

30 **end**

代码第28行，我们把cmos\_v\_pixel的值由原先的272改为256，相应的我们的sdram\_max\_addr也要改动，数值上等于cmos\_h\_pixel\* cmos\_v\_pixel。这些值将传到摄像头驱动模块，从而去配置摄像头的图像输出。

在这里我们有必要对以上的改动做出解释：本实验我们的目标是要兼容五种RGB LCD屏幕大小，首先我们要去配置摄像头输出的图像大小，使图像大小和对应的屏幕大小相等，另一方面，为了在我们原有的SDRAM控制架构的基础上实现本次实验，我们需要满足一个条件：摄像头输出图像的像素值个数大小是512（也突发长度）的整倍数，对于4.3寸LCD屏大小为480\*272，如果图像大小也设置为480\*272显然无法满足，为了解决这个问题，我们将图像纵向像素值个数减小了16，图像大小改成了480\*256，这样一来我们将可以沿用上个实验的SDRAM控制的代码。

接下来我们介绍LCD顶层模块（lcd），我们先给出该模块的顶层代码：

1 **module** lcd**(**

2 **input** clk **,** //lcd模块输入时钟

3 **input** rst\_n **,** //lcd模块复位，低电平有效

4 **input** **[**12**:**0**]** rd\_h\_pixel **,** //图像横向分辨率

5 //RGB LCD接口

6 **output** lcd\_hs **,** //LCD 行同步信号

7 **output** lcd\_vs **,** //LCD 场同步信号

8 **output** lcd\_de **,** //LCD 数据输入使能

9 **inout** **[**15**:**0**]** lcd\_rgb **,** //LCD RGB565颜色数据

10 **output** lcd\_bl **,** //LCD 背光控制信号

11 **output** lcd\_rst **,** //LCD 复位信号

12 **output** lcd\_pclk**,** //LCD 采样时钟

13 **output** clk\_lcd **,**

14 **input** **[**15**:**0**]** rd\_data **,** //像素点数据

15 **output** rd\_en **,** //请求像素点颜色数据输入

16 **output** **[**15**:**0**]** ID\_lcd //LCD的ID

17 **);**

18

19 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20 //\*\* main code

21 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

22

23 //RGB565数据输出

24 **assign** lcd\_rgb **=** lcd\_de **?** pixel\_data **:** 16'dz**;**

25 **wire** **[**10**:**0**]** pixel\_xpos**;**

26 **wire** **[**10**:**0**]** pixel\_ypos**;**

27 **wire** data\_req**;**

28 **wire** **[**15**:**0**]** pixel\_data**;**

29

30 //读rgb lcd ID 模块

31 rd\_id u\_rd\_id**(**

32 **.**clk **(**clk**)** **,** //lcd模块输入时钟

33 **.**rst\_n **(**rst\_n**)** **,** //lcd模块复位，低电平有效

34

35 **.**lcd\_rgb **(**lcd\_rgb**),** //LCD屏传过来的RGB信号

36 **.**ID\_lcd **(**ID\_lcd**)** //读出的LCD ID

37 **);**

38

39 //分频模块，根据不同的LCD ID输出相应的频率的驱动时钟

40 clk\_div u\_clk\_div**(**

41 **.**clk **(**clk**)** **,** //lcd模块输入时钟

42 **.**rst\_n **(**rst\_n**)** **,** //lcd模块复位，低电平有效

43

44 **.**ID\_lcd **(**ID\_lcd**),** //LCD的ID

45 **.**clk\_lcd **(**clk\_lcd**)** //产生的lcd驱动时钟

46 **);**

47

48 //实现字符叠加

49 lcd\_disply u\_lcd\_disply**(**

50

51 **.**lcd\_clk **(**clk\_lcd**)** **,** //lcd显示驱动时钟

52 **.**sys\_rst\_n **(**rst\_n**)** **,** //lcd模块复位，低电平有效

53 //RGB LCD接口

54 **.**pixel\_xpos **(**pixel\_xpos**),** //像素点横坐标

55 **.**pixel\_ypos **(**pixel\_ypos**),** //像素点纵坐标

56 **.**ID\_lcd **(**ID\_lcd**)** **,** //LCD的ID

57 **.**rd\_data **(**rd\_data**)** **,** //摄像头采集到的图像数据

58 **.**rd\_h\_pixel **(**rd\_h\_pixel**),** //图像横向分辨率

59 **.**pixel\_data **(**pixel\_data**)** //像素点数据,

60 **);**

61

62 //lcd驱动模块

63 lcd\_driver u\_lcd\_driver**(**

64 **.**lcd\_clk **(**clk\_lcd**)** **,** //lcd驱动时钟

65 **.**sys\_rst\_n **(**rst\_n**)** **,** //lcd模块复位，低电平有效

66

67 **.**lcd\_hs **(**lcd\_hs**)** **,** //lcd行信号

68 **.**lcd\_vs **(**lcd\_vs**)** **,** //lcd场信号

69 **.**lcd\_de **(**lcd\_de**)** **,** //LCD数据输入使能

70 **.**lcd\_bl **(**lcd\_bl**)** **,** //LCD 背光控制信号

71 **.**lcd\_rst **(**lcd\_rst**)** **,** //LCD 复位信号

72 **.**lcd\_pclk **(**lcd\_pclk**)** **,** //LCD 采样时钟

73

74 **.**data\_req **(**rd\_en**)** **,** //像素请求信号

75 **.**pixel\_xpos **(**pixel\_xpos**),** //像素点横坐标

76 **.**pixel\_ypos **(**pixel\_ypos**),** //像素点纵坐标

77 **.**ID\_lcd **(**ID\_lcd**)** //LCD的ID

78 **);**

79

80 **endmodule**

以上主要是LCD顶层模块的顶层例化，从代码中我们可以看到LCD顶层模块一共包含四个模块，同时可以看到各信号的定义。代码第24行做了一个判断，lcd\_de为高则把像素值赋给RGB，否则RGB为高阻态，即不显示任何图像。

关于四个模块，我们只对lcd\_disply模块以及lcd\_driver模块进行讲解。另外两个模块未作改动，具体代码讲解，可以参看“OV5640摄像头RGB TFT-LCD显示”实验。

先来看lcd\_driver模块。

本实验，lcd\_driver模块沿用了开拓者“OV5640摄像头RGB TFT-LCD显示”实验原有模块，并在原有代码上只做了一处修改，我们只贴出改动部分的代码：

109 //请求像素点坐标输入

110 **assign** data\_req\_valid**=(((**cnt\_h **>=**h\_sync**+**h\_back**-**1'b1**)** **&&** **(**cnt\_h **<** h\_sync**+**h\_back**+**h\_disp**-**1'b1**))**

111 **&&** **((**cnt\_v **>=** v\_sync**+**v\_back**)** **&&** **(**cnt\_v **<** v\_sync**+**v\_back**+**v\_disp**)))**

112 **?** 1'b1 **:** 1'b0**;**

113

114 //请求像素点图像数据输入

115 **assign** data\_req **=** **(**ID\_lcd **==**ID\_4342**)** **?** **(**data\_req\_valid **&&** pixel\_ypos **>** 16**):** data\_req\_valid **;**

代码第110行，我们定义了data\_req\_valid信号，该信号实际上就是之前的模块的data\_req信号，即请求图像像素输入。如前文所述，本实验为了能继续沿用之前的SDRAM控制架构，对于4.3寸480\*272屏，我们要对其图片大小进行修改，我们将图像纵向像素尺寸少16，这样造成的结果是：图片像素的大小是填不满整个屏幕的，于是我们做了相应的处理。

代码第115行，我们对LCD的ID进行判断，如果是4.3寸480\*272屏，需要在像素纵坐标大于16以后才让它有效，我们让像素请求信号延迟16行发出，这意味着对于该屏幕前16行的像素点位置将无数据填充，也就是让前屏幕前16行“空”出来。而屏幕的16行以后数据请求信号开始有效，这样一来图片像素的大小刚好可以填满剩下的屏幕。其他屏幕则不需要要这样处理。

最后讲解lcd\_disply模块，以下是该模块全部代码：

1 **module** lcd\_disply**(**

2 **input** lcd\_clk**,** //lcd模块驱动时钟

3 **input** sys\_rst\_n**,** //复位信号

4 //RGB LCD接口

5 **input** **[** 10**:**0**]** pixel\_xpos**,** //像素点横坐标

6 **input** **[** 10**:**0**]** pixel\_ypos**,** //像素点纵坐标

7 **input** **[**15**:**0**]** ID\_lcd **,** //LCD的ID

8 **input** **[**15**:**0**]** rd\_data**,** //读侧FIFO0的输出数据

9 **input** **[**12**:**0**]** rd\_h\_pixel**,** //摄像头输出的水平方向分辨率

10 **output** **reg** **[**15**:**0**]** pixel\_data //像素点数据,

11 **);**

12

13 //LCD的ID

14 **parameter** ID\_4342 **=** 0**;**

15 **parameter** ID\_7084 **=** 1**;**

16 **parameter** ID\_7016 **=** 2**;**

17 **parameter** ID\_1018 **=** 5**;**

18 **parameter** ID\_4384 **=** 4**;**

19 //颜色定义

20 **localparam** RED **=** 16'b11111\_000000\_00000**;** //字符颜色

21 **localparam** BLUE **=** 16'b00000\_000000\_11111**;** //字符区域背景色

22 **localparam** BLACK **=** 16'b00000\_000000\_00000**;** //屏幕背景色

23 //字符定义

24 **reg** **[**63**:**0**]** char0**[**15**:**0**];** //字符数组0

25 **reg** **[**63**:**0**]** char1**[**15**:**0**];** //字符数组1

26 **reg** **[**127**:**0**]** char2**[**32**:**0**];** //字符数组2

27 **reg** **[**127**:**0**]** char3**[**32**:**0**];** //字符数组3

28 //给字符数组0的赋值：OV5640 1 (16\*64)

29 **always** **@(posedge** lcd\_clk**)** **begin**

30 char0**[**0**]** **<=** 64'h0000000000000000 **;**

31 char0**[**1**]** **<=** 64'h0000000000000000 **;**

32 char0**[**2**]** **<=** 64'h0000000000000000 **;**

33 char0**[**3**]** **<=** 64'h38E77E1804180008 **;**

34 char0**[**4**]** **<=** 64'h444240240C240038 **;**

35 char0**[**5**]** **<=** 64'h824240400C420008 **;**

36 char0**[**6**]** **<=** 64'h8244404014420008 **;**

37 char0**[**7**]** **<=** 64'h8224785C24420008 **;**

38 char0**[**8**]** **<=** 64'h8224446224420008 **;**

39 char0**[**9**]** **<=** 64'h8228024244420008 **;**

40 char0**[**10**]** **<=** 64'h822802427F420008 **;**

41 char0**[**11**]** **<=** 64'h8218424204420008 **;**

42 char0**[**12**]** **<=** 64'h4410442204240008 **;**

43 char0**[**13**]** **<=** 64'h3810381C1F18003E **;**

44 char0**[**14**]** **<=** 64'h0000000000000000 **;**

45 char0**[**15**]** **<=** 64'h0000000000000000 **;**

46 **end**

47

48 //给字符数组1的赋值: OV5640 2 (16\*64)

49 **always** **@(posedge** lcd\_clk**)** **begin**

50 char1**[**0**]** **<=** 64'h0000000000000000 **;**

51 char1**[**1**]** **<=** 64'h0000000000000000 **;**

52 char1**[**2**]** **<=** 64'h0000000000000000 **;**

53 char1**[**3**]** **<=** 64'h38E77E180418003C **;**

54 char1**[**4**]** **<=** 64'h444240240C240042 **;**

55 char1**[**5**]** **<=** 64'h824240400C420042 **;**

56 char1**[**6**]** **<=** 64'h8244404014420042 **;**

57 char1**[**7**]** **<=** 64'h8224785C24420002 **;**

58 char1**[**8**]** **<=** 64'h8224446224420004 **;**

59 char1**[**9**]** **<=** 64'h8228024244420008 **;**

60 char1**[**10**]** **<=** 64'h822802427F420010 **;**

61 char1**[**11**]** **<=** 64'h8218424204420020 **;**

62 char1**[**12**]** **<=** 64'h4410442204240042 **;**

63 char1**[**13**]** **<=** 64'h3810381C1F18007E **;**

64 char1**[**14**]** **<=** 64'h0000000000000000 **;**

65 char1**[**15**]** **<=** 64'h0000000000000000 **;**

66 **end**

67

68 //给字符数组2的赋值: OV5640 1 (32\*128)

69 **always** **@(posedge** lcd\_clk**)** **begin**

70 char2**[**0**]** **<=** 128'h00000000000000000000000000000000**;**

71 char2**[**1**]** **<=** 128'h00000000000000000000000000000000**;**

72 char2**[**2**]** **<=** 128'h00000000000000000000000000000000**;**

73 char2**[**3**]** **<=** 128'h00000000000000000000000000000000**;**

74 char2**[**4**]** **<=** 128'h00000000000000000000000000000000**;**

75 char2**[**5**]** **<=** 128'h00000000000000000000000000000000**;**

76 char2**[**6**]** **<=** 128'h03C07C1E0FFC01E0006003C000000080**;**

77 char2**[**7**]** **<=** 128'h0C30180C0FFC06180060062000000180**;**

78 char2**[**8**]** **<=** 128'h1818180810000C1800E00C3000001F80**;**

79 char2**[**9**]** **<=** 128'h100818081000081800E0181800000180**;**

80 char2**[**10**]** **<=** 128'h300C1808100018000160181800000180**;**

81 char2**[**11**]** **<=** 128'h300C0C10100010000160180800000180**;**

82 char2**[**12**]** **<=** 128'h60040C10100010000260300C00000180**;**

83 char2**[**13**]** **<=** 128'h60060C10100030000460300C00000180**;**

84 char2**[**14**]** **<=** 128'h60060C1013E033E00460300C00000180**;**

85 char2**[**15**]** **<=** 128'h60060C20143036300860300C00000180**;**

86 char2**[**16**]** **<=** 128'h60060620181838180860300C00000180**;**

87 char2**[**17**]** **<=** 128'h60060620100838081060300C00000180**;**

88 char2**[**18**]** **<=** 128'h60060620000C300C3060300C00000180**;**

89 char2**[**19**]** **<=** 128'h60060640000C300C2060300C00000180**;**

90 char2**[**20**]** **<=** 128'h60060340000C300C4060300C00000180**;**

91 char2**[**21**]** **<=** 128'h20060340000C300C7FFC300C00000180**;**

92 char2**[**22**]** **<=** 128'h300C0340300C300C0060180800000180**;**

93 char2**[**23**]** **<=** 128'h300C0380300C180C0060181800000180**;**

94 char2**[**24**]** **<=** 128'h10080180201818080060181800000180**;**

95 char2**[**25**]** **<=** 128'h1818018020180C1800600C3000000180**;**

96 char2**[**26**]** **<=** 128'h0C30010018300E3000600620000003C0**;**

97 char2**[**27**]** **<=** 128'h03C0010007C003E003FC03C000001FF8**;**

98 char2**[**28**]** **<=** 128'h00000000000000000000000000000000**;**

99 char2**[**29**]** **<=** 128'h00000000000000000000000000000000**;**

100 char2**[**30**]** **<=** 128'h00000000000000000000000000000000**;**

101 char2**[**31**]** **<=** 128'h00000000000000000000000000000000**;**

102 **end**

103

104 //给字符数组3的赋值: OV5640 2 (32\*128)

105 **always** **@(posedge** lcd\_clk**)** **begin**

106 char3**[**0**]** **<=** 128'h00000000000000000000000000000000**;**

107 char3**[**1**]** **<=** 128'h00000000000000000000000000000000**;**

108 char3**[**2**]** **<=** 128'h00000000000000000000000000000000**;**

109 char3**[**3**]** **<=** 128'h00000000000000000000000000000000**;**

110 char3**[**4**]** **<=** 128'h00000000000000000000000000000000**;**

111 char3**[**5**]** **<=** 128'h00000000000000000000000000000000**;**

112 char3**[**6**]** **<=** 128'h03C07C1E0FFC01E0006003C0000007E0**;**

113 char3**[**7**]** **<=** 128'h0C30180C0FFC06180060062000000838**;**

114 char3**[**8**]** **<=** 128'h1818180810000C1800E00C3000001018**;**

115 char3**[**9**]** **<=** 128'h100818081000081800E018180000200C**;**

116 char3**[**10**]** **<=** 128'h300C180810001800016018180000200C**;**

117 char3**[**11**]** **<=** 128'h300C0C1010001000016018080000300C**;**

118 char3**[**12**]** **<=** 128'h60040C10100010000260300C0000300C**;**

119 char3**[**13**]** **<=** 128'h60060C10100030000460300C0000000C**;**

120 char3**[**14**]** **<=** 128'h60060C1013E033E00460300C00000018**;**

121 char3**[**15**]** **<=** 128'h60060C20143036300860300C00000018**;**

122 char3**[**16**]** **<=** 128'h60060620181838180860300C00000030**;**

123 char3**[**17**]** **<=** 128'h60060620100838081060300C00000060**;**

124 char3**[**18**]** **<=** 128'h60060620000C300C3060300C000000C0**;**

125 char3**[**19**]** **<=** 128'h60060640000C300C2060300C00000180**;**

126 char3**[**20**]** **<=** 128'h60060340000C300C4060300C00000300**;**

127 char3**[**21**]** **<=** 128'h20060340000C300C7FFC300C00000200**;**

128 char3**[**22**]** **<=** 128'h300C0340300C300C0060180800000404**;**

129 char3**[**23**]** **<=** 128'h300C0380300C180C0060181800000804**;**

130 char3**[**24**]** **<=** 128'h10080180201818080060181800001004**;**

131 char3**[**25**]** **<=** 128'h1818018020180C1800600C300000200C**;**

132 char3**[**26**]** **<=** 128'h0C30010018300E300060062000003FF8**;**

133 char3**[**27**]** **<=** 128'h03C0010007C003E003FC03C000003FF8**;**

134 char3**[**28**]** **<=** 128'h00000000000000000000000000000000**;**

135 char3**[**29**]** **<=** 128'h00000000000000000000000000000000**;**

136 char3**[**30**]** **<=** 128'h00000000000000000000000000000000**;**

137 char3**[**31**]** **<=** 128'h00000000000000000000000000000000**;**

138 **end**

139

140 //显示逻辑判断

141 **always@(\*)** **begin** //判断是否为4.3寸480\*272屏，且纵坐标是位于字符区域

142 **if(**ID\_lcd **==**ID\_4342 **&&** pixel\_ypos **>=** 0 **&&** pixel\_ypos **<** 17 **)begin**

143 //判断像素坐标是否在左半边屏幕中间

144 **if(**pixel\_xpos **>=** 88 **&&** pixel\_xpos **<** 152 **)begin**

145 //读取字模OV5640 1(16\*64)

146 **if(**char0**[**pixel\_ypos**][**63**-(**pixel\_xpos **-** 88**)])**

147 pixel\_data **=**BLUE**;**//字模数组中的“1”显示蓝色

148 **else**

149 pixel\_data **=** 0**;** //字模数组中的“0”显示黑色

150 **end** //判断像素坐标是否在右半边屏幕中间

151 **else** **if(**pixel\_xpos **>=** 328 **&&** pixel\_xpos **<** 392 **)begin**

152 //读取字模OV5640 2(16\*64)

153 **if(**char1**[**pixel\_ypos**][**63**-(**pixel\_xpos **-** 328**)])**

154 pixel\_data **=**BLUE**;**//字模数组中的“1”显示蓝色

155 **else**

156 pixel\_data **=** 0**;** //字模数组中的“0”显示黑色

157 **end**

158 **else**

159 pixel\_data**=**0**;** //纵坐标位于字符区域内的字符两边区域显示黑色

160 **end** //判断是4.3寸480\*272以外的屏，且纵坐标是否位于字符区域

161 **else** **if** **(**ID\_lcd **!=**ID\_4342 **&&** pixel\_ypos **>=** 0 **&&** pixel\_ypos **<** 32**)begin**

162 //判断像素坐标是否在左半边屏幕中间

163 **if(**pixel\_xpos **<** **(**rd\_h\_pixel**[**12**:**2**]+**64**)**

164 **&&** pixel\_xpos **>=** **(**rd\_h\_pixel**[**12**:**2**]-**64**)** **)begin**

165 //读取字模OV5640 1 (32\*128)

166 **if(**char2**[**pixel\_ypos**][**127**-(**pixel\_xpos**-**rd\_h\_pixel**[**12**:**2**]+**64**)])**

167 pixel\_data **=**BLUE**;**//字模数组中的“1”显示蓝色

168 **else** //字模数组中的“0”显示图像像素值

169 pixel\_data **=** rd\_data**;**

170 **end** //判断像素坐标是否在右半边屏幕中间

171 **else** **if(**pixel\_xpos **<** **(**rd\_h\_pixel**[**12**:**2**]\***3**+**64**)**

172 **&&** pixel\_xpos **>=** **(**rd\_h\_pixel**[**12**:**2**]\***3**-**64**))begin**

173 //读取字模OV5640 2 (32\*128)

174 **if(**char3**[**pixel\_ypos**][**63**-**pixel\_xpos**+(**rd\_h\_pixel**[**12**:**2**])\***3**])**

175 pixel\_data **=**BLUE**;**//字模数组中的“1”显示蓝色

176 **else** //字模数组中的“0”显示图像像素值

177 pixel\_data **=** rd\_data**;**

178 **end**

179 **else** //纵坐标位于字符区域内的字符两边区域显示黑色

180 pixel\_data **=** rd\_data**;**

181 **end**

182 **else**

183 pixel\_data **=** rd\_data**;** //所有屏幕纵坐标位于字符区域外时显示像素值

184 **end**

185

186 **endmodule**

本模块实现了五种RGB LCD屏不同区域显示内容的逻辑判断，同时实现了字符叠加功能。

代码第14行到22行，定义了五种屏幕以及不同颜色对应的参数。代码第29行到138行给我们要用到的四个字符的模值进行赋值。对于4.3寸480\*272的LCD屏幕，我们选择大小为16\*64的字符，左右半边分别叠加上的“OV5640 1”和“OV5640 2”，其他LCD屏幕，我们选择大小为32\*128的字符，叠加的字符内容一样。

代码从139行往后是显示逻辑的具体实现，由于代码涉及到了判断条件之间的多重嵌套，为了让我们快速理清思路，我们结合代码制作了如图 1.4.6所示的显示逻辑图。结合代码和图，我们来具体介绍。

首先我们对屏幕类型和图像像素纵坐标进行判断，对于所有屏幕，如果纵坐标值不在字符显示区域内，我们执行代码第176行，让屏幕显示图像像素的值。只有当纵坐标值位于字符显示区域内，我们正式开始字符叠加处理。

代码第141行，通过ID判断此时连接的屏幕是否是4.3寸480\*272屏，同时判断了像素点纵坐标的位置。代码第144行，此时开始判断像素点横坐标位置，如果此时横坐标处于左半边的中间位置，此时执行代码第146行，开始读取“OV5640 1”的字符数组的值。代码第147到149行，将字模数组中为“1”的点的像素值赋值为蓝色，为“0”的点像素值赋值为黑色。这就是4.3寸480\*272屏左半边字符叠加的实现。代码第151到157行，实现的是4.3寸480\*272屏右半边字符叠加，实现的代码形式一模一样，只是字模数组变为了“OV5640 2”，在这里我们不再重复。代码第159行，字符区域非中间位置，显示黑色。

代码第161行往后是对于非4.3寸480\*272屏的字符叠加处理。代码实现形式和4.3寸480\*272屏的是一样的。对具体内容而言，存在不同之处：首先是所叠加的字符的大小不同，当然，字符叠加的位置范围也相应地做了修改。其次是对于字模数组中为“0”的点的像素值赋值不同，前者为黑色，后者为图片像素值，字符区域非中间位置的值赋值也不同，同样前者为黑色，后者为图片像素值。

对于以上字符叠加这两种不同的处理，我们有必要在这里解释：前文所知我们把4.3寸480\*272屏的图像大小配置成了480\*256，这样造成的结果是，显示时我们的图像数据将无法将屏幕“填满”，会空出16行，因此我们考虑用固定颜色，黑色来填充。而对于其他屏幕我们的图像大小配置成屏幕大小，是刚好填满的，所以我们直接赋具体的像素值。在实验现象中表现为4.3寸480\*272屏字符背景为黑色，而其他屏幕背景则时图像本身。



图 1.4.6 显示逻辑

## 下载验证

首先我们打开实验工程dual\_ov5640\_lcd，在工程所在，目录下打开dual\_ov5640\_lcd/par文件夹，在里面找到“dual\_ov5640\_lcd.qpf”并双击打开。注意工程所在的路径名只能由字母、数字以及下划线组成，不能出现中文、空格以及特殊字符等。工程打开后如图 1.5.1所示：

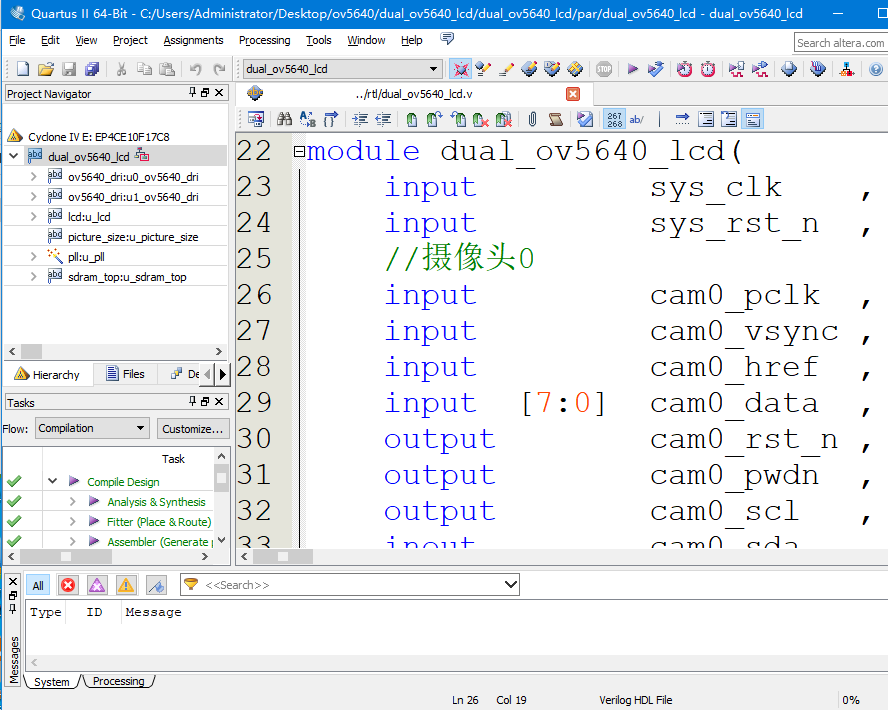


图 1.5.1 双目OV5640摄像头LCD显示工程

工程打开后通过点击工具栏中的“Programmer”图标打开下载界面，通过“Add File”按钮选择双目摄像头工程dual\_ov5640\_lcd/par/output\_files目录下的“dual\_ov5640\_lcd.sof”文件。在下载验证之前我们需要连接好硬件，将双目OV5640摄像头模块连接P6扩展口，注意镜头方向朝外，如下图所示：

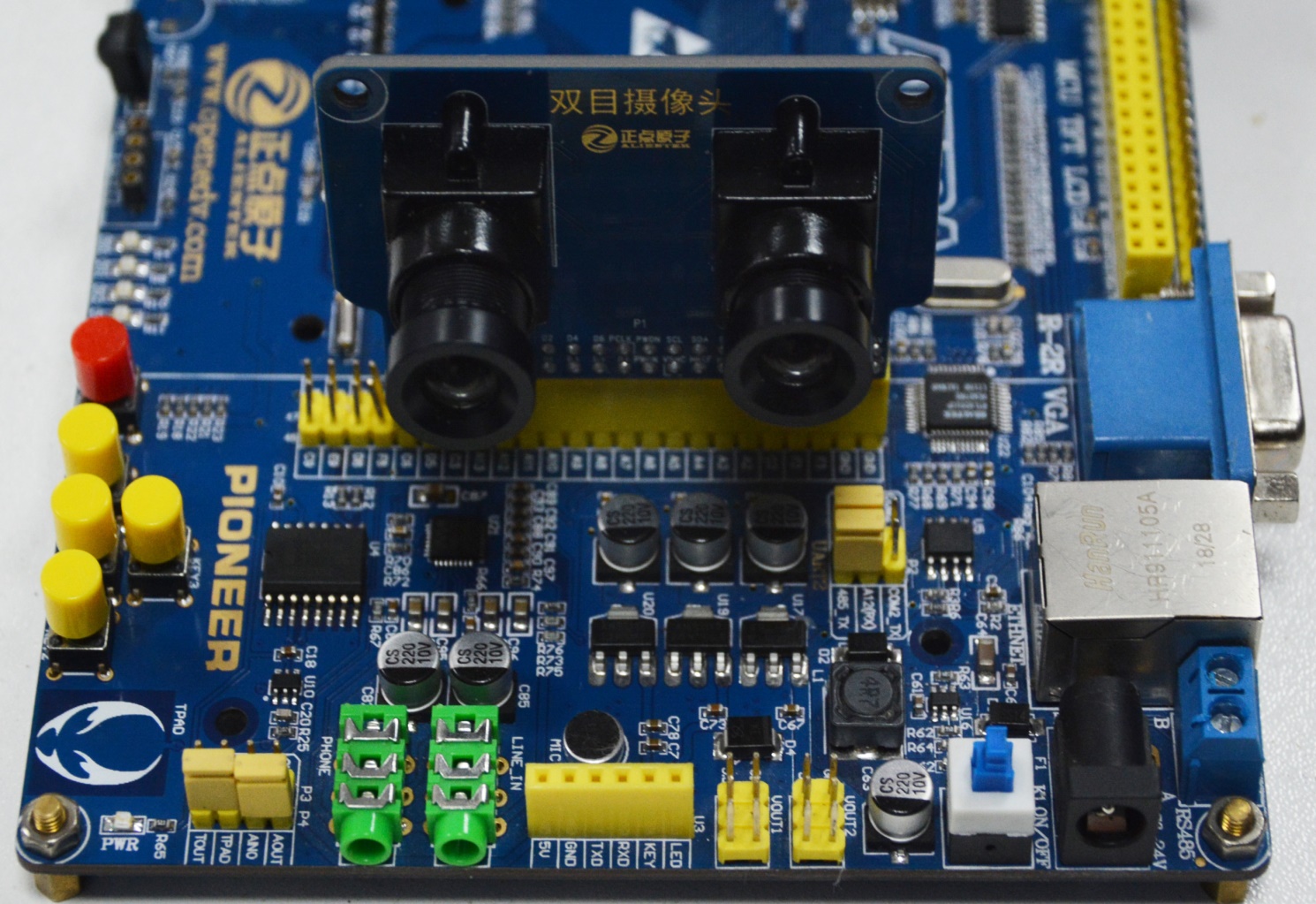


图 . 双目摄像头接插实物图

将下载器一端连电脑，另一端与开发板上JTAG端口连接，然后连接VGA显示屏，连接电源线并打开电源开关。

回到下载界面，点击“Hardware Setup”，在弹出的对话框中选择当前的硬件连接“USB-Blaster[USB-0]”。然后点击“Start”将工程编译完成后得到的sof文件下载到开发板中，如图 1.5.3，下载成功。

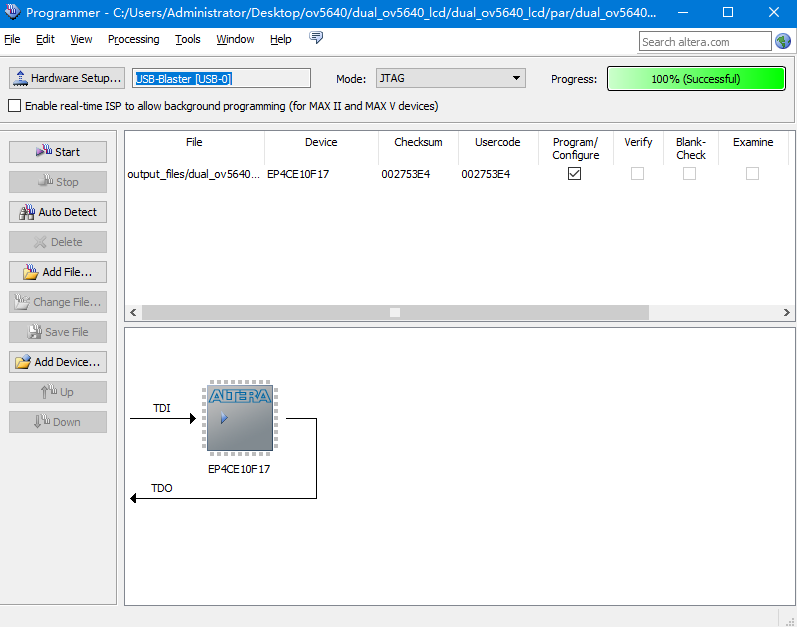


图 1.5.3 程序下载完成界面

下载完成后，我们可以看到双目摄像头在LCD屏幕上显示的图片，如下图所示：

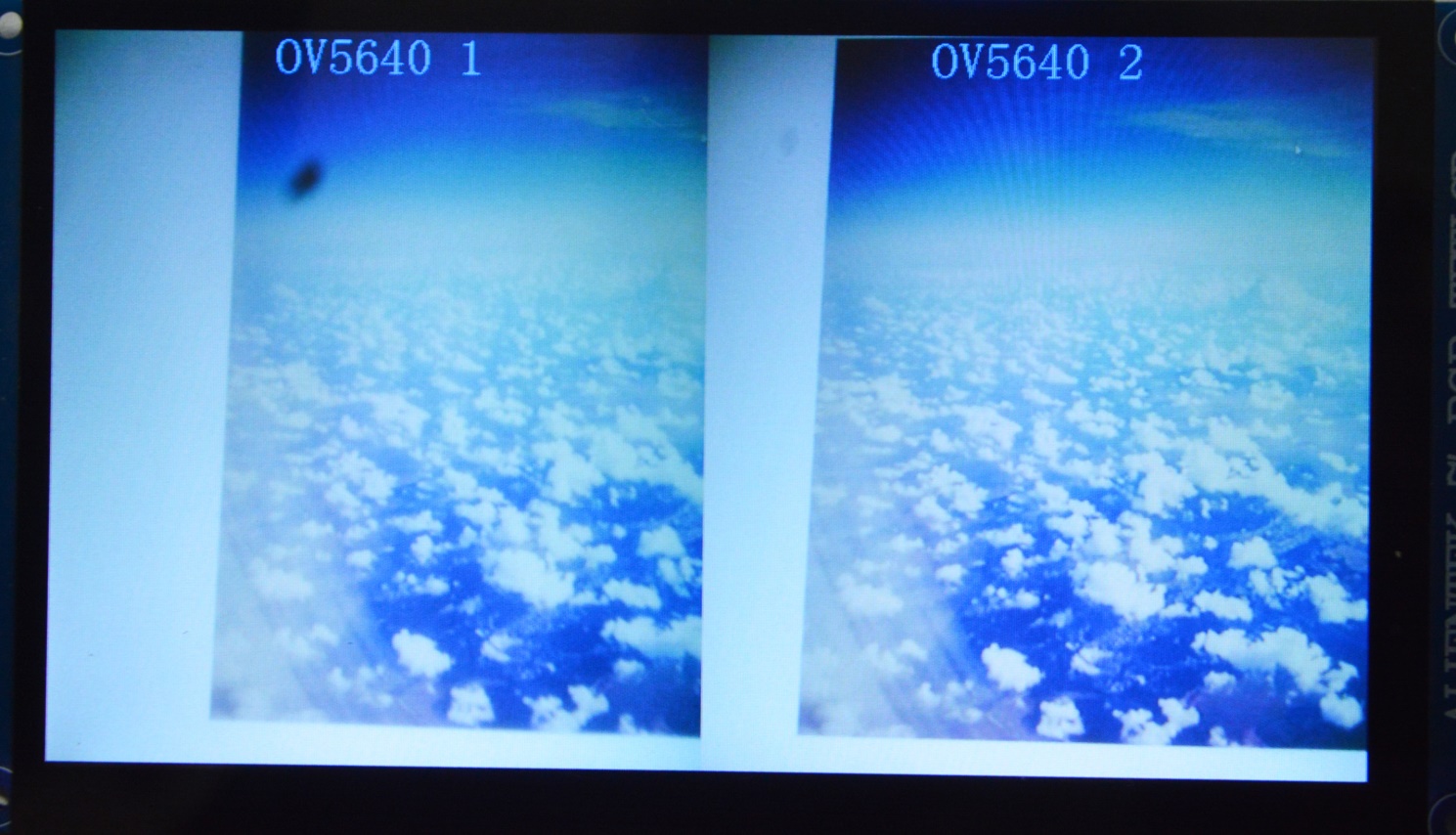


图 1.5.4 VGA显示屏显示效果