2019 年第三届全国大学生 FPGA 创新设计大赛

紫光同创赛道

移形换影 -- --基于 PGL22G 的图像旋转控制器

队 伍 名 称：旗开得胜

参 赛 平 台：紫光 DS002—1 Logos 系列

参 赛 队 员：15140220090 巩文红

17140210068 吴家桢

17020150102 马迪

目录

第一部分 设计概述 (Design Introduction) 1

1.1 设计目的.....1

1.2 设计要求.....1

第二部分 平台介绍(Platform Introduction).....3

2.1 简介.....3

2.2 FPGA.....4

2.3FPGA 供电系统.....4

2.4 有源晶振.....4

2.5 DDR3.....5

2.6 HDMI.....5

2.7 串口.....6

第三部分 基础模块介绍(Basic module design).....9

3.1 OV5640 摄像头的初始化.....9

3.2 DDR.....10

3.3 乒乓操作.....11

3.4 VGA 时序.....12

3.5 HDMI 显示.....15

3.6 黑白以及灰度显示.....16

第四部分 具体模块设计(Module Design).....17

4.1 图像缓存设计.....17

4.2 AXI 总线数据传输方案.....21

4.3 DDR 的数传输控制.....24

4.4 坐标对应关系.....29

4.5 ddr+串口联合测试.....30

4.6 图像旋转方案.....35

4.7 图像旋转计算.....37

4.8 Python 编写上位机.....46

第五部分,效果展示(Results show).....49

5.1 原图显示(正常模式)..... 49

5.2 旋转模式.....49

5.3 平移模式.....51

5.4 缩放模式.....52

5.5 灰度模式.....53

5.6 黑白模式.....53

第六部分，总结(Summary).....55

第一部分 设计概述 (Design Introduction)

1.1 设计目的

随着各类图像旋转算法的层出不穷，图像旋转逐渐成为近年来各类赛事的热门赛题。然而在基于 FPGA 的图像旋转设计方面，可行的方案较少。因此，我们本次采用了紫光同创的 PGL22G 这块开发板进行图像旋转方案的设计，制作成了一个完整的具有快速处理，实时显示的系统。

本作品从图像旋转这一经典的问题出发，采用 CORDIC(Coordinate Rotation Digital Computer)算法，结合图传技术，实时显示技术，以 FPGA 作为核心处理器，通过自制的上位机软件实现软件对硬件的精确控制，达到对摄像头采集的图像进行实时旋转并且显示的目的，并且可以通过上位机对旋转后的图像进行显示模式，灰度阈值的设定。

本设计的核心思路为：在图像旋转设计中，插入一个图像旋转模块。将从摄像头缓存的图像先读取出来，组合成一帧旋转的图像后再写入 ddr 中，再由显示驱动模块读取进行显示。

1.2 设计要求

PGL22G + DDR + CMOS Sensor + HDMI TX

功能描述：

系统主要实现视频任意角度旋转（360°，最小角度 1°）

1、CMOS Sensor 将图像以 DVP/MIPI 信号形式传到 FPGA；

2、FPGA 通过外部控制，**实现任意角度（精度 1 度）旋转控制**；外部控制可自己选择，如 UART、12C、GPIO、按键等；

3、FPGA 内部接 DDR 控制器，实现图像旋转缓存；

4、图像旋转处理完成后，通过 HDMI 输出到显示器显示。

1.3 作品效果

我们团队的作品不仅实现了比赛预定的视频图像旋转的功能，同时我们团队还另外完成了视频图像的平移，简单缩放，灰度显示，黑白显示等功能。具体的展示效果详见第五章的效果展示。

第二部分 平台介绍(Platform Introduction)

2.1 简介

PGL22G(核心板型号，下同)核心板，是紫光同创公司开发的 Logos 系列 FPGA 高性能核心板，具有高速，高带宽，高容量等特点，适合高速数据通信，视频图像处理，高速数据采集等方面使用。

这款核心板使用了 1 片 Micron 公司 MT41J128M16HA-125 这款 DDR3 芯片，容量为 256MB； DDR3 芯片和 FPGA 芯片总线宽度为 16bit，数据时钟频率高达 800Mhz；这样的配置，可以满足高带宽的数据处理的需求。板上的 128Mb QSPI FLASH 芯片的型号为 W25Q128，用于存储 FPGA 系统的启动文件。

这款核心板扩展出 114 个 FPGA 的 IO 口(默认 3.3V 电平标准)，其中有 40 个 IO 可以通过修改核心板上的 LDO 芯片来改变电平标准。对于需要大量 IO 的用户，此核心 板将是不错的选择。而且，FPGA 芯片到接口之间走线做了等长和 12 对 LVDS 差分走线处理，并且核心板尺寸仅为 45*55（mm），对于二次开发来说非常适合。

PGL22G 核心板正面图

2.2 FPGA

前面已经介绍过了，我们所使用的 FPGA 型号为 PGL22G6CMBG324，属于紫光同创公司的 Logos 系列产品，速度等级为-6，温度等级为商业级 C。此型号为 MBG324 封装，324 个引脚。Logos 系列 FPGA 命名规则如图所示。

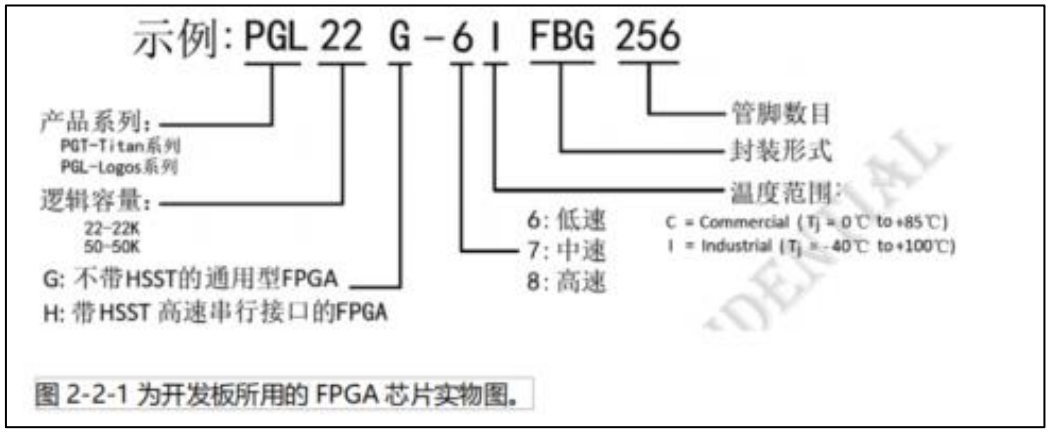


图 2-2-1 为开发板所用的 FPGA 芯片实物图。

2.3FPGA 供电系统

紫光同创 Logos FPGA 电源有 VCCINT，VCCIO L0，VCCIO L1，VCCIO L2，VCCIO R0，VCCIO R1，VCCIO R2，VCCAUX 和 VCCIO。VCCINT 为 FPGA 内核供电引脚， 需接+1.1V；VCCAUX 为 FPGA 辅助供电引脚，接 3.3V；VCCIO 为 FPGA 的各个 BANK 的电压，包含 BANK L0~L2，BANK R0~R2，在 PGL22G 核心板上，BANK L1，BANK L2 因为需要连接 DDR3，BANK 的电压连接的是 1.5V，其它 BANK 的电压都是 3.3V，其中 BANK R2 的 VCCIO 由可选择的两路 LDO 供电，可以通过跳电阻更改 BANK 的电 平。

2.4 有源晶振

PGL22G 核心板上配有一个 50Mhz 的有源晶振，用于 FPGA 的系统主时钟。晶振输出连接到 FPGA 的时钟输入管脚(Pin B5)，这个时钟可以用来驱动 FPGA 内的用户逻辑

电路，用户可以通过配置 FPGA 内部的 PLLs 来实现更高的时钟。

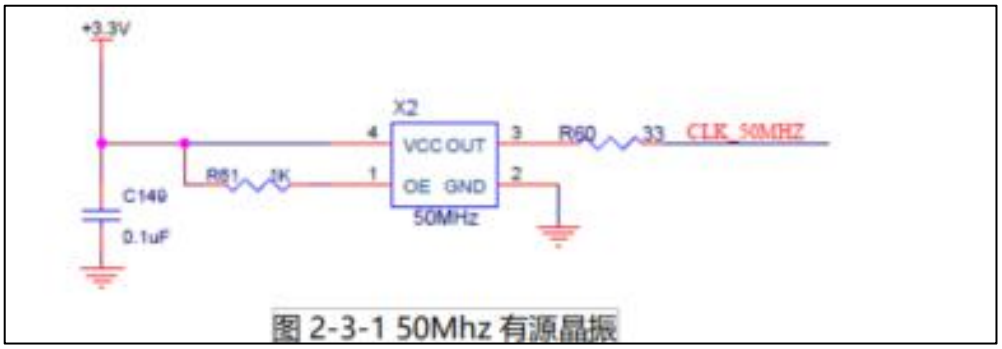


图 2-3-1 50Mhz 有源晶振

2.5 DDR3

PGL 核心板上配有 1 个 Micron(美光）的 256MB 的 DDR3 芯片，型号为 MT41J128M16HA-125。DDR 的总线宽度共为 16bit。DDR3 ddr 的最高运行时钟速度可达 400MHz。该 DDR3 存储系统直接连接到了 FPGA 的 BANK L1 和 BANK L2 的存储器接口上。

图2-4-2 DDR3 DRAM实物图

2.6 HDMI

HDMI 输出接口的实现，是通过 FPGA 的 4 路 LVDS 差分信号（3 路数据和一路时钟）接口直接驱动 HDMI 输出，为开发板提供不同格式的视频输出接口。

其中，HDMI 接口和 FPGA 之间的 LVDS 差分信号的连接使用 AC Couple 的模式，

起到隔直的左右。另外在硬件设计上，每对 LVDS 差分信号上增加了 TVS 保护管，防止外面静电对 FPGA 的损坏。HDMI 输出接口的硬件连接如图 3-3-1 所示。

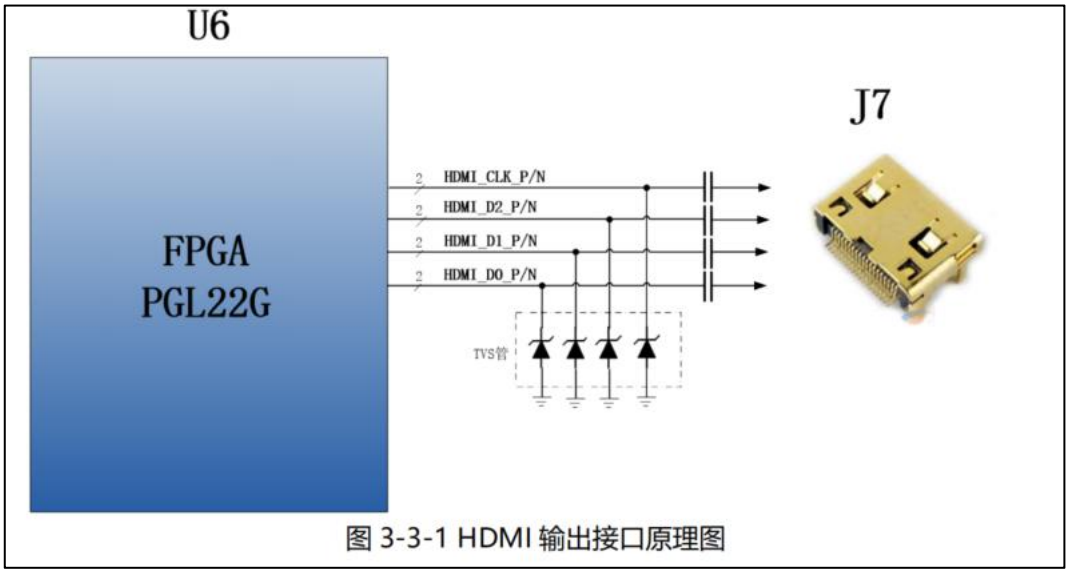
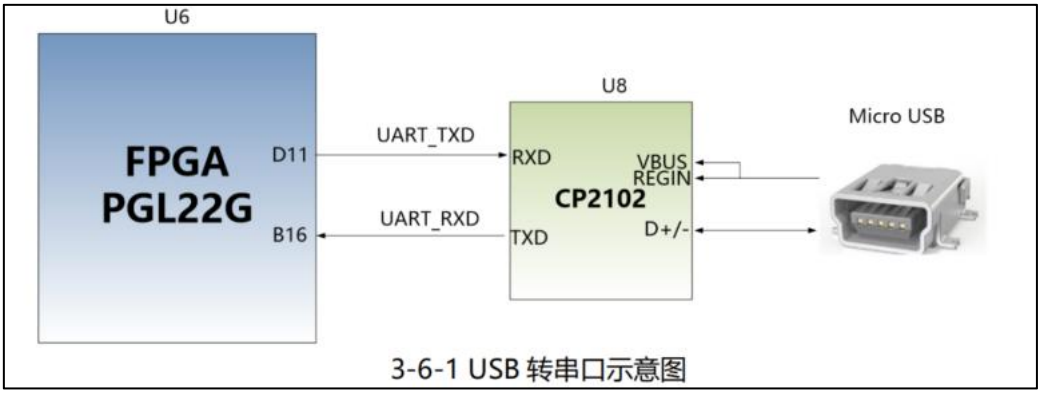


图 3-3-1 HDMI 输出接口原理图

2.7 串口

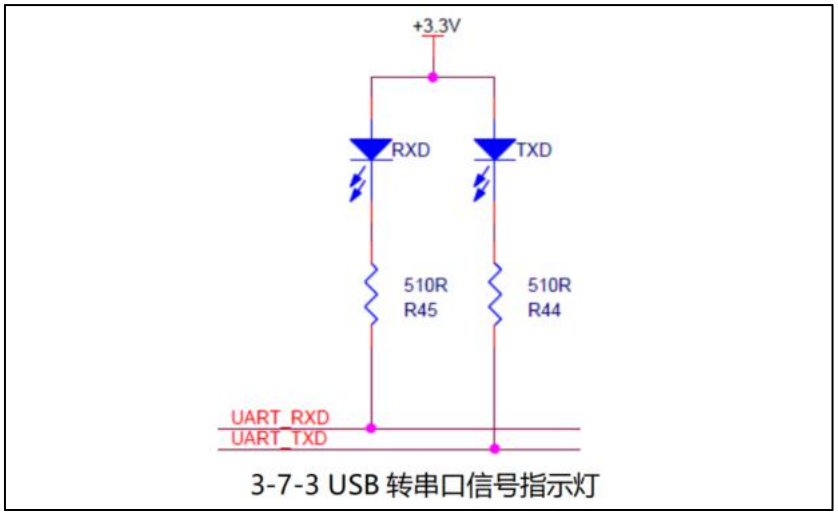
PGL22G 开发板包含了 Silicon Labs CP2102GM 的 USB-UAR 芯片，USB 接口采用 MINI USB 接口，可以用一根 USB 线将它连接到上 PC 的 USB 口进行串口数据通信 。USB Uart 电路设计的示意图如下图所示:



3-6-1 USB 转串口示意图

同时对串口信号设置了 2 个 PCB 上丝印为 TXD 和 RXD 的 LED 指示灯， TXD 和

RXD LED 灯会指示串口是否有数据发出或者是否有数据接受，如下图所示，



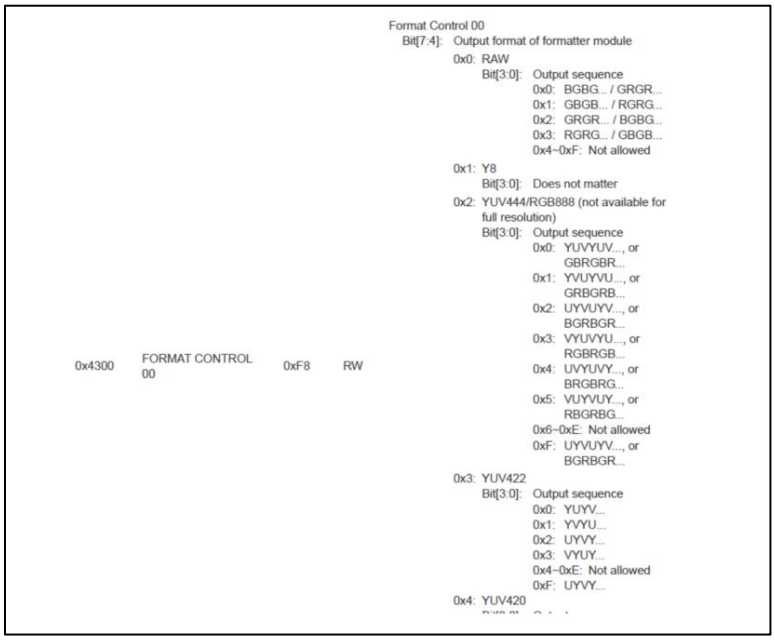
3-7-3 USB 转串口信号指示灯

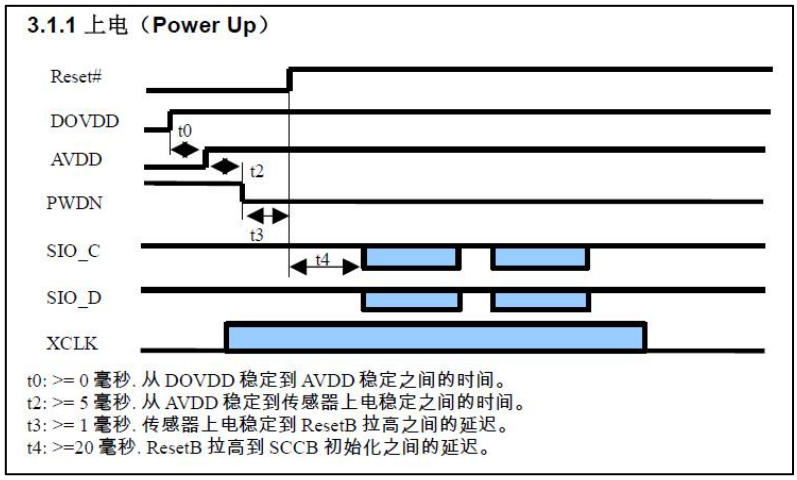
第三部分 基础模块介绍(Basic module design)

3.1 OV5640 摄像头的初始化

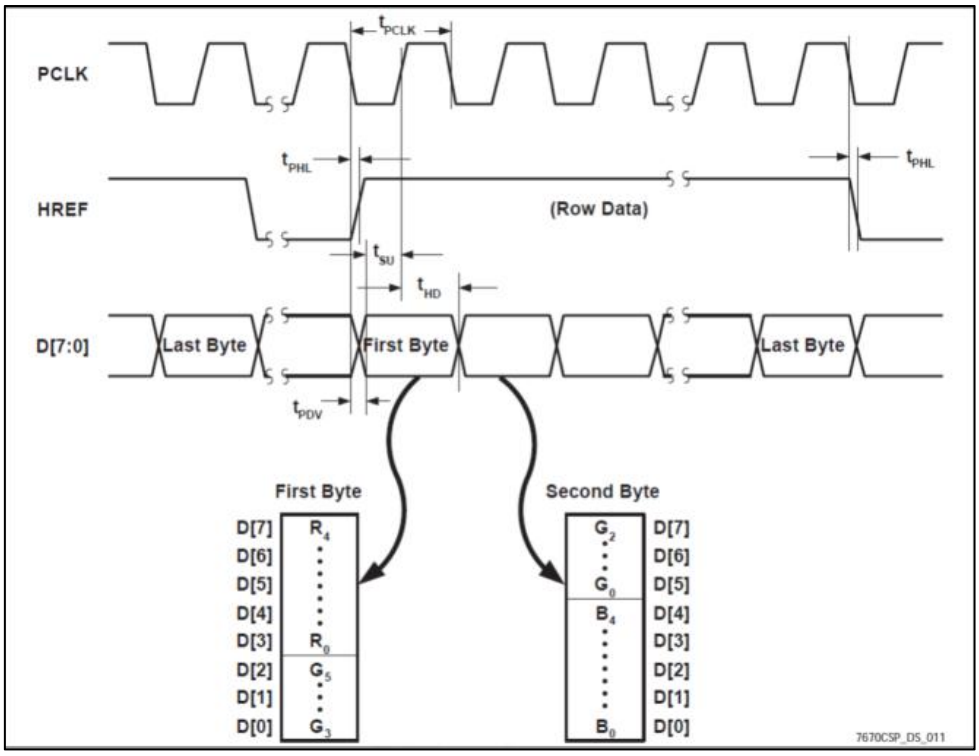
OV5640 摄像头模组采用美国 OmniVision(豪威)CMOS 芯片图像传感器 OV5640，支持自动对焦的功能。OV5640 芯片支持 DVP 和 MIPI 接口。

OV5640 的寄存器配置是通过 FPGA 的 I2C（也称为 SCCB 接口）接口来配置。用户需要配置正确的寄存器值让 OV5640 输出我们需要的图像格式。



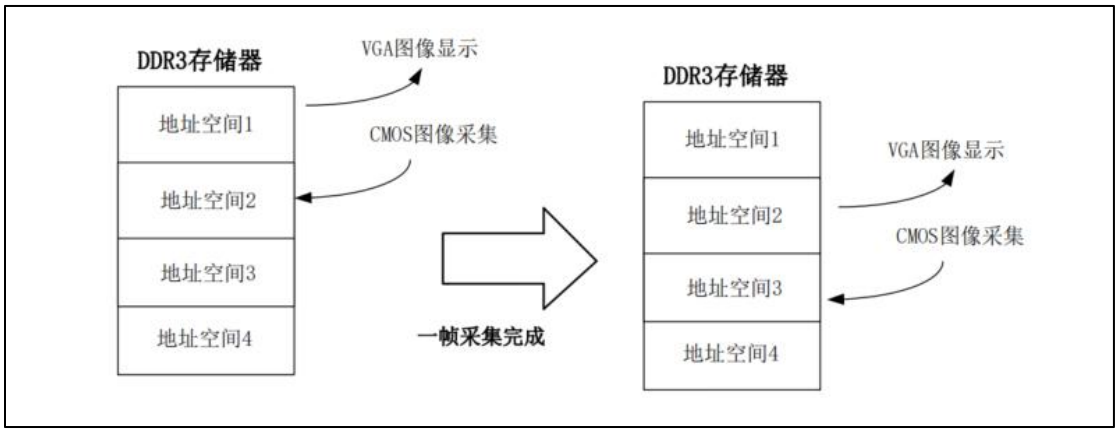


OV5640 在 HREF 信号为高时输出一行的图像数据，输出数据在 PCLK 的上升沿的时候有效。因为 RGB565 显示每个像数为 16bit，但 OV5640 每个 PCLK 输出的是 8bit,所以每个图像的像数分两次输出，第一个 Byte 输出为 R4~R0 和 G5~G3， 第二个 Byte 输出为 G2~G0 和 B4~B0,将前后 2 个字节拼接起来就是 16Bit RGB565 数据。



3.2 DDR

DDR 作为数据传输过程中的重要节点，在摄像头采菊数据的传输方面扮演者关键的角色，DDR 对数据的读于写都直接的影响摄像头的数据传输的稳定性。如下图是 ddr 对摄像头数据的采集过程：



3.3 乒乓操作

乒乓操作是一个主要用于数据流控制的处理技巧，典型的乒乓操作如图所示：

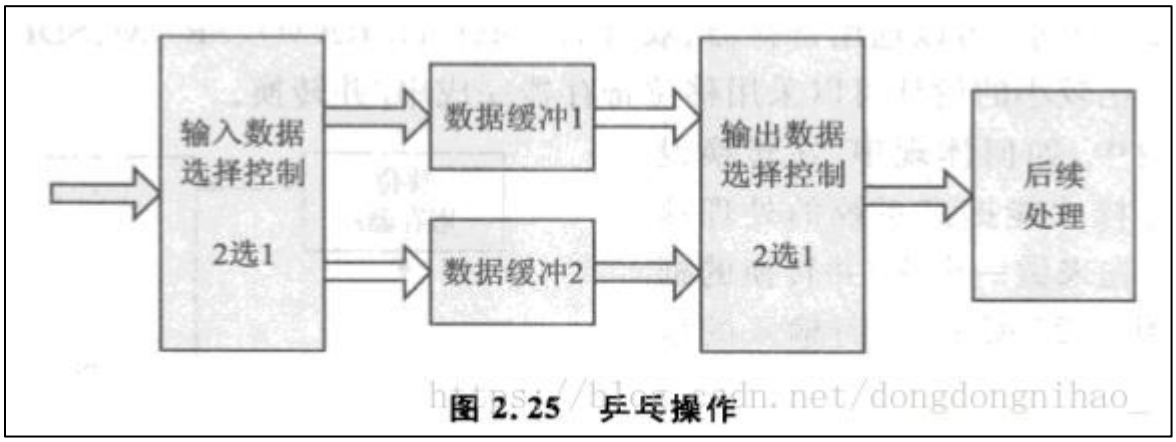


图 2.25 乒乓操作

外部输入数据流通过“输入数据选择控制”模块送入两个数据缓冲区中，数据缓冲模块可以为任何存储模块，比较常用的存储单元为双口 RAM（Dual RAM），SRAM，SDRAM，FIFO 等。

在第 1 个缓冲周期，将输入的数据流缓存到“数据缓冲 1”模块，在第 2 个缓冲周期，“输入数据选择控制”模块将输入的数据流缓存到“数据缓冲 2”模块的同时，“输出数据选择控制”模块将“数据缓冲 1”模块第一个周期缓存的数据流送到“后续处理”，模块进行后续的数据处理，在第三个缓冲周期，在“输入数据选择控制”模块的

再次切换后，输入的数据流缓存到“数据缓冲 1”模块，与此同时，“输出数据选择控制”模块也做出切换，将“数据缓冲 2”模块缓存的第二个周期的数据送到“后续处理模块”，如此循环。

这里正是利用了乒乓操作完成数据的无缝缓冲与处理，乒乓操作可以通过“输入数据选择控制”和“输出数据选择控制”按节拍，相互配合地进行来回切换，将经过缓冲的数据流没有停顿的送到“后续处理模块”。

比如将乒乓操作运用在液晶显示的控制模块上，如图所示。

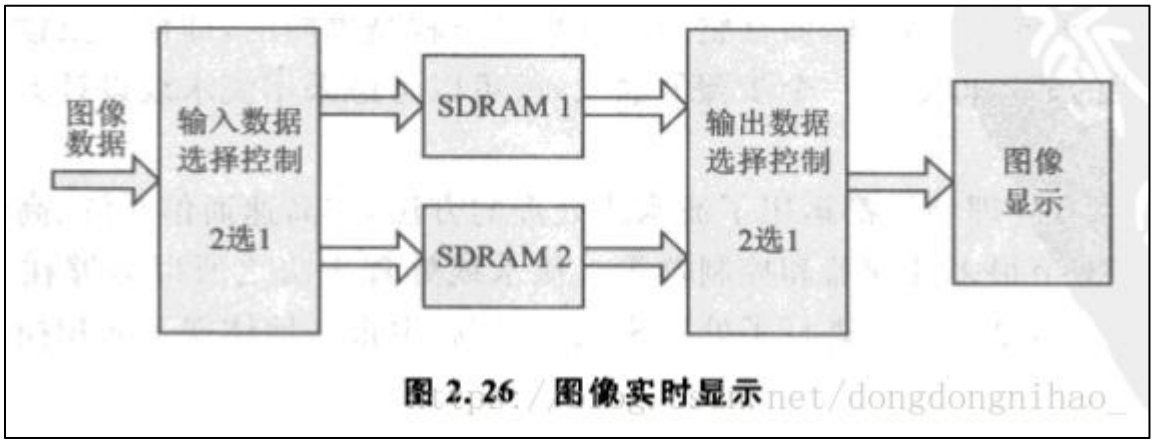


图 2.26 图像实时显示

对于外部接口传输的图像数据，以一帧图像为单位进行 SDRAM 的切换控制，当 SDRAM1 缓存图像数据时，液晶显示的是 SDRAM2 的数据图像；反之，当 SDRAM2 缓存图像数据时，液晶显示的是 SDRAM1 的数据图像，如此反复，这样出路的好处在于液晶显示图像切换瞬间完成，掩盖了可能比较缓慢的图像数据流变换过程。

3.4 VGA 时序

显示器扫描方式分为逐行扫描和隔行扫描：逐行扫描是扫描从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。隔行扫描是指电子束扫描时每隔一行扫一行，完成一屏后再返回来扫

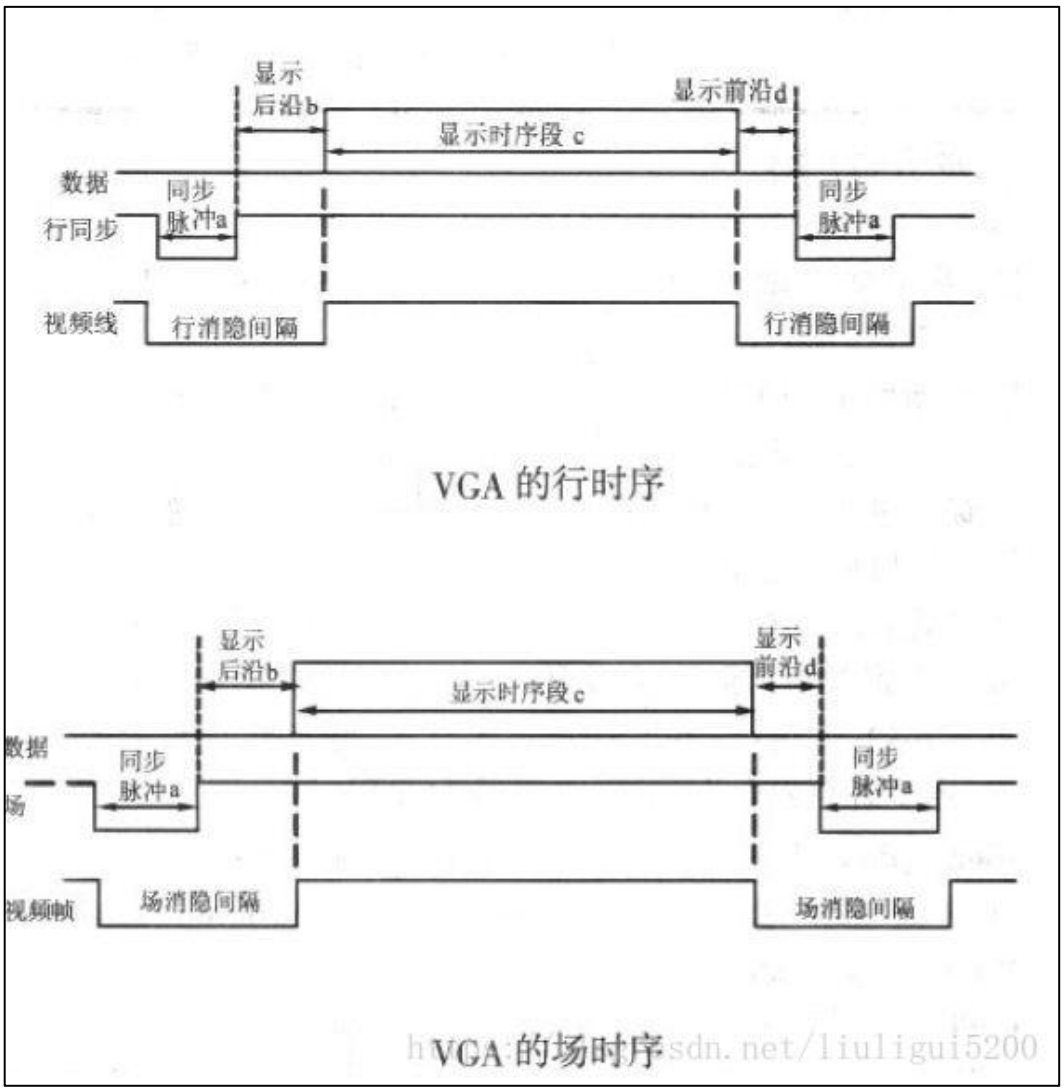
描剩下的行，隔行扫描的显示器闪烁的厉害，会让使用者的眼睛疲劳。

完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即刷新一屏的频率，常见的有 60Hz，75Hz 等等。标准的 VGA 显示的场频 60Hz,行频 31.5KHz。

行场消隐信号:是针对老式显像管的成像扫描电路而言的。电子枪所发出的电子束从屏幕的左上角开始向右扫描，一行扫完需将电子束从右边移回到左边以便扫描第二行。在移动期间就必须有一个信号加到电路上，使得电子束不能发出。不然这个回扫线会破坏屏幕图像的。这个阻止回扫线产生的信号就叫作消隐信号,场信号的消隐也是一个道理。

显示带宽:带宽指的显示器可以处理的频率范围。如果是 60Hz 刷新频率的 VGA,其带宽达 640x480x60=18.4MHz,70Hz 的刷新频率 1024x768 分辨率的 SVGA,其带宽达 1024x768x70=55.1MHz。

时钟频率：以 640x480@59.94Hz(60Hz)为例，每场对应 525 个行周期(525 = 10 + 2 + 480+33),其中 480 为显示行。每场有场同步信号,该脉冲宽度为 2 个行周期的负脉冲，每显示行包括 800 点时钟,其中 640 点为有效显示区,每一行有一个行同步信号，该脉冲宽度为 96 个点时钟。由此可知：行频为 525*59.94=31469Hz,需要点时钟频率：525*800*59.94 约 25Mhz.

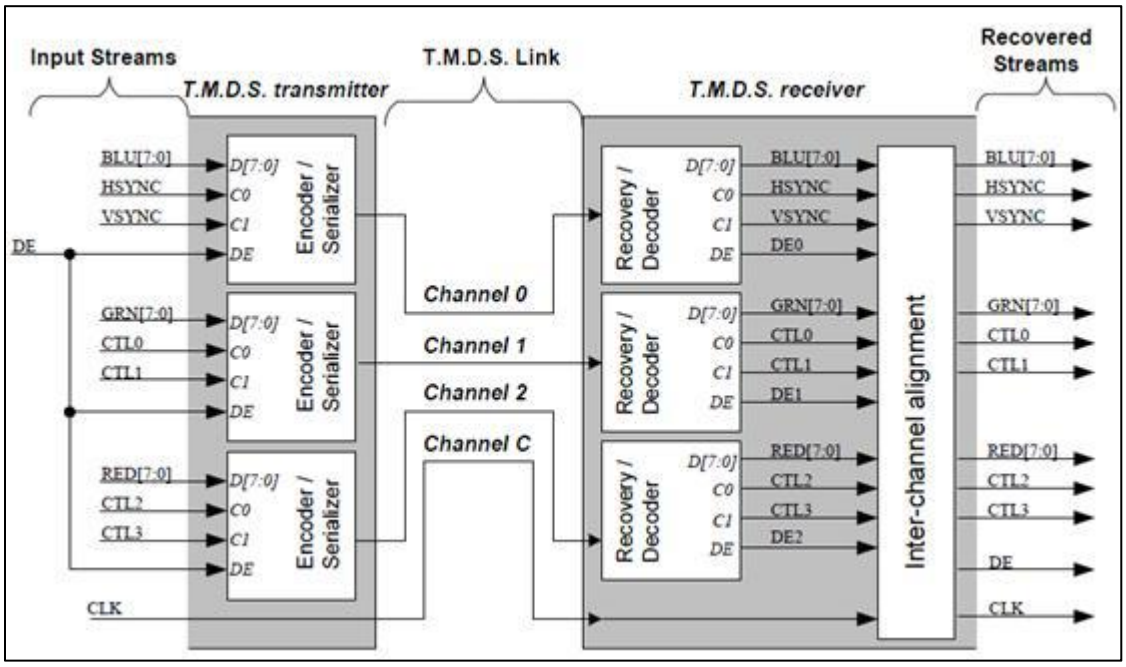


由 VGA 的行时序可知：每一行都有一个负极性行同步脉冲（Sync a），是数据行的结束标志，同时也是下一行的开始标志。在同步脉冲之后为显示后沿（Back porch b），在显示时序段（Display interval c）显示器为亮的过程，RGB 数据驱动一行上的每一个像素点，从而显示一行。在一行的最后为显示前沿（Front porch d）。在显示时间段(Display interval c)之外没有图像投射到屏幕是插入消隐信号。同步脉冲(Sync a)、显示后沿(Back porch b)和显示前沿(Front porch d)都是在行消隐间隔内(Horizontal Blanking Interval)，当消隐有效时，RGB 信号无效，屏幕不显示数据。

3.5 HDMI 显示

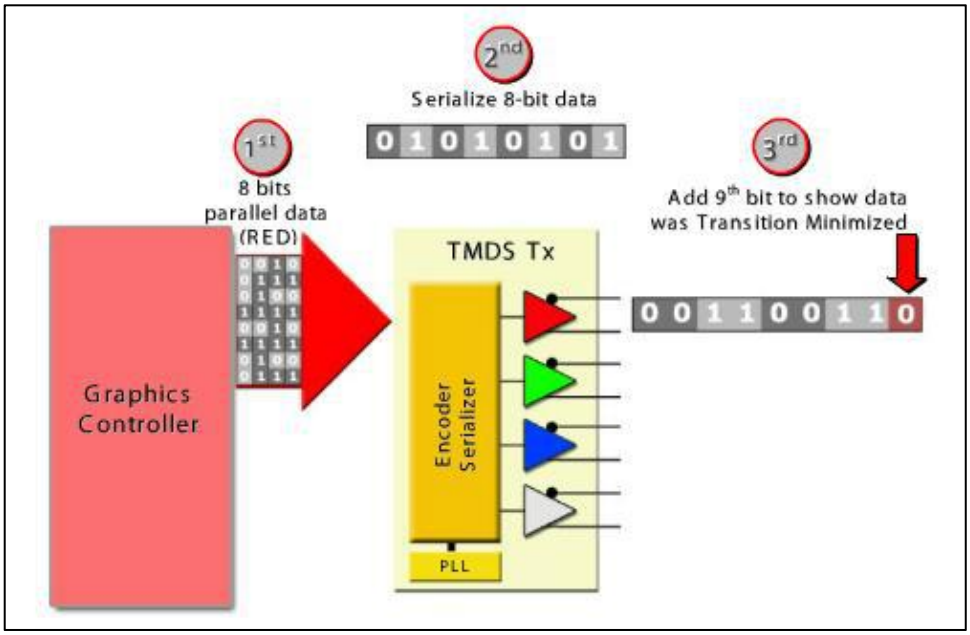
HDMI 采用和 DVI 相同的传输原理——TMDS（Transition Minimized Differential signal），最小化传输差分信号。

TMDS 传输系统分为分为两个部分：发送端和接收端。TMDS 发送端收到 HDMI 接口传来的表示 RGB 信号的 24 位并行数据（TMDS 对每个像素的 RGB 三原色分别按 8bit 编码，即 R 信号有 8 位，G 信号有 8 位，B 信号有 8 位），然后对这些数据进行编码和并/串转换，再将表示 3 个 RGB 信号的数据分别分配到独立的传输通道发送出去。接收端接收来自发送端的串行信号，对其进行解码和串/并转换，然后发送到显示器的控制端。与此同时也接收时钟信号，以实现同步。



8 位数据经过编码和直流平衡得到 10 位最小化数据，这仿佛增加了冗余位，对传输链路的带宽要求更高，但事实上，通过这种算法得到的 10 位数据在更长的同轴电缆中传输的可靠性增强了。

下图是一个例子，说明对一个 8 位的并行 RED 数据编码、并/串转换。



3.6 黑白以及灰度显示

在原图显示的基础上，我们添加了黑白以及灰度的显示模式，提高了整个系统的可观赏性，下面具体分析算法如下。

首先是灰度算法，此算法有一个著名的心理学公式引出：

Gray = R*0.299 + G*0.587 + B*0.114。

但在运算的时候由于公式中出现了浮点数，因此为了提高运算速度，则需要采用整数算法。注意到系数都是 3 位精度的没有，我们可以将它们缩放 1000 倍来实现整数运算算法：Gray = (R*299 + G*587 + B*114 + 500) / 1000。

第四部分 具体模块设计(Module Design)

4.1 图像缓存设计

本文讲述下利用 ddr 缓存从摄像头处得到的数据，并将图像显示到显示屏上的工程架构。注：本文不涉及具体的代码讲解，只描述其中的实现思路。

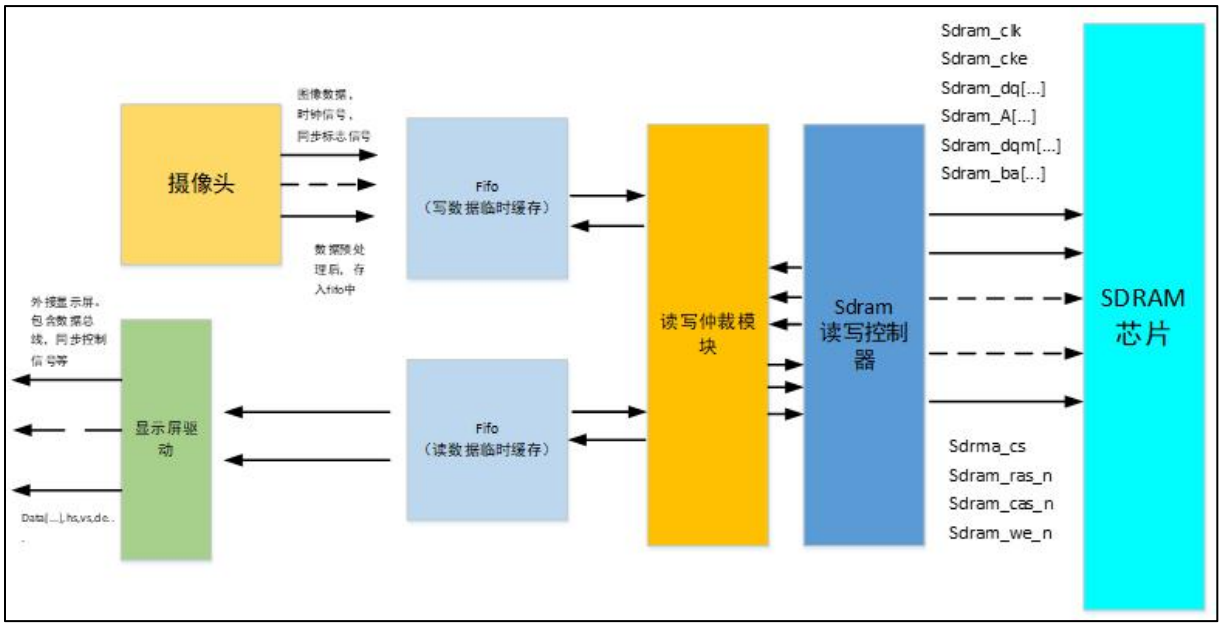
该工程有如下模块组成：

摄像头数据捕获模块，

读 fifo，写 fifo，

读写仲裁模块，

ddr 控制器



每个模块的作用如下。

1，摄像头数据捕获模块。

一般摄像头传输的数据，一个像素是由相邻两个时钟的数据拼接而成的，在数据捕

获模块中，要根据同步控制信号，将两个时钟的数据拼接在一起输出。

2，读 fifo，写 fifo。

用于临时缓存读写数据，解决数据的跨时钟域问题。

3，读写仲裁模块。

用于判断和控制合适从 ddr 中读写数据。

4，ddr 控制器。

用来实现 ddr 的初始化配置，读写时序的实现。

该工程的总体设计概要图如下，从摄像图获取的的数据先临时缓存到写 fifo 中，然后再写入到 ddr 中，需要显示图像时，图像数据先从 ddr 临时储存到读 fifo 中，然后显示屏驱动模块读取 fifo 中的数据，并将图像显示到显示屏上。在 ddr 中，将图像数据按照从左到右，从上到下的顺序储存在一片连续地址的储存区域内，便可以方便突发读写传输。

1，为什么要用 ddr。

由于摄像头产生的数据时钟速率和显示屏的时钟不匹配，而且当摄像头传来数据时，显示屏驱动模块不一定在此时需要数据，况且同一时刻摄像头传进的像素的位置不一定就是显示屏正刷新到的位置。故不能将摄像头的数据直接直接传输到显示屏驱动模块。

一张 24bit 的的全彩图的，每个像素点占据 3byte 的空间,常见的不同尺寸的图片一帧所占空间如下：

1080*1920*3byte=6,220,800byte=5,9mbyte;

960*480*3byte=1，382，400byte=1.3mbyte

480*272*3byte=391,680byte=0.3mbyte

通常板载的 ddr 的大小有 16MB,32MB，64MB，128MB，等等，由设计成本决定，但用于储存一帧图片绰绰有余。

只有将图像数据临时储存起来，这样读写便互不所影响。当摄像头传来有效数据时，便将数据存入 ddr，当显示屏需要数据时，便从 ddr 中读取数据，发送给显示屏用于显示。这样便可以将读写储存隔离。

低成本的板载 ddr 的时钟频率也在 100MHz 以上，数据宽度为 16bit,数据带宽最小为 100M*2byte（16bit）=200Mbyte/s。按 60HZ，640*480 的图片计算，一帧图片要进行读和写两次操作，通常由摄像头传入的图像为 16 位宽，那么图像传输的速率为：640 * 480 * 2btye(16bit) * 2 * 60 = 70Mbyte,可见 ddr 的带宽足够，满足图像实时显示要求。

2，为什么要用 fifo 做数据的临时缓存?

为什么不能将数据直接存入 ddr，还要经过 fifo 临时缓存呢?

一是由于跨时钟域。摄像头产生的数据的速率和 ddr 的写入时钟速率不匹配。一般而言，ddr 的时钟频率都要大于像素时钟的一到两倍。故需要一个读写双时钟的 fifo。将像素时钟接入到 fifo 的写时钟端口，像素数据有效信号接入到 fifo 的写使能端口，将有效的数据写入到 fifo 中。将 ddr 的时钟接到 fifo 的读时钟端口，用于从 fifo 中读取数据。

二是由于 ddr 的突发传输，要保证连续突发传输时每一个数据都是有效数据。如果直接将摄像头的数据传输到 ddr 端口 ，在连续传输的 128 个字节中，像素数据不一定

是全部连续有效的数据。只将有效数据储存进 fifo 中后，从 fifo 中读出的连续数据一定都是连续的有效数据。

对于读 fifo，也是同理，显示屏的驱动时钟和 ddr 的时钟也存在跨时钟域，而且在突发读取时，也不能保证从 ddr 中传出的数据正好和显示所需要 Dev 数据相匹配。故需要有一个双时钟端口的用于读数据的 fifo 做数据的临时缓存。

3，何时将 fifo 中的数据写入 ddr?

ddr 的突发长度设置为 128，设置当写 fifo 中的数据量大于 128 时，进行一次写突发传输，将 fifo 中的 128 个数据搬移到 ddr 中。图像数据不断从摄像头传入，写进 fifo 中，每当写入的数据大于预设值，就进行一次写突发传输。这个有读写仲裁模块所控制。

4，何时将 ddr 中的数据读出

显示屏驱动模块要从读 fifo 中读取数据，设置一个阈值，当读 fifo 中的数据小于该阈时，便进行一次突发传输，将数据从 ddr 中临时缓存到 fifo 中，用于显示驱动的读操作。

5，何为乒乓操作，为何要乒乓操作?

乒乓操作的具体总法为，在 ddr 中设置两个储存空间，用于储存两张图片。分别用于储存摄像头传来的图像数据，和显示屏读取数据。一帧图片传输完成后，读写区域互换。这样能保证在显示屏显示时，能够显示一张完整的图像。如果利用同一片储存区域来储存图像，当前一帧图像还没有读取显示完成，下一帧图像的数据就将该区域覆盖，那么显示屏上的画面会出现拖影现象，两帧图像会交叠在一起。

4.2 AXI 总线数据传输方案

4.2.1 总体设计

经过摄像头获取到图像数据后，经过图像数据的预处理后，将图像数据缓存到 ddr 中。显示模块再从 DDR 中读取图像数据，显示到显示屏上。

图像数据通过 AXI 接口写入到 DDR 中，通过 AXI 总线从 DDR 中读取。这期间要跨三个时钟域。分别是 摄像头数据输出时钟，AXI 读写时钟，显示屏驱动时钟。在跨时钟域传输数据时，数据都要经过 fifo 缓存。

分别设置两个 fifo 模块，用于跨时钟域的数据缓存。一个设置在摄像头数据模块和 DDR 中，用于缓存向 DDR 中写入的数据。一个设置在 DDR 和显示屏驱动模块之间，用于缓存从 DDR 中读取出的数据。

在将数据从摄像头传输到 DDR 中时，当 fifo 中的数据数量大于预设值，启动一次突发写传输，将缓存在 fifo 中的数据缓存到 DDR 的预设的地址中。

在将数据从 DDR 传输到显示控制模块中时，当 fifo 中的数据数量小于预设值，启动一次突发读传输，从 DDR 的预设的地址中读取出图像数据，缓存到 fifo 中，共显示模块读取。

4.2.2 数据的写入方案

AXI 的写传输由 fifo 中缓存的数据数量触发，当储存在 fifo 中的数据大于预设值时，进行一次突发传输，将缓存在 fifo 中的数据写入到 DDR 中。

当开始启动一次突发传输时，先发送突发传输的起始地址，地址信号通过 axi_awaddr 端口传输。同时将 axi_awvalid 信号置高，用与通知主机地址通道有有效

的地址信息。从机通过 AXI_AWREADY 信号来标志接收到该地址信息。主机接收到 AXI_AWREADY 有效信号后，将 axi_awvalid 置低，直到下一次突发传输，再重复该过程。

启动一次突发传输时，便将 axi_wvalid 置为有效，同时将第一个要发送的数据放置在 axi_wdata 总线上。AXI 写通道的数据（axi_wdata）要和 axi_wvalid 对齐，从机通过 AXI_WREADY 来标识从机开始接收数据，待从机的 AXI_WREADY 有效时，表示从机已准备好接受数据，在 AXI_WREADY 有效的后，从机在下一个时钟寄存接收该数据。此时主机便可将 AXI_WREADY 作为读使能信号，在下一个时钟输出突发传输的下一个数据。重复该步骤，直到一次突发传输完成。即在 AXI_WREADY 有效时，主机在下一个时钟发送新数据，从机在下一时钟寄存该时钟接收到的数据。

主机通过 axi_wlast 信号来标记一次突发传输的最后一个数据。该信号由主机产生，通过设置一个计数器来记录突发传输的数据的数量。 axi_wlast 信号要提前一个时钟判断使能，和要发送的最后一个数据对齐。当计数值到倒数第二个数据时，使能 axi_wlast 信号，使之在下一个时钟有效，这样便可以将 axi_wlast 信号和最后一个发送的数据对齐。

在紫光的官方 fifo 中，数据输出段不会直接输出当前读指针指向的数据，需要提供一个读使能信号后，数据才会在下一时钟在读数据端口输出。故在启动突发传输时，也要提供一个时钟的读 fifo 使能信号，将第一个要写入的数据从 fifo 中读出。在从机的 AXI_WREADY 有效后，从机开始接受数据，主机也要从 fifo 中读出新的数据送给从机。

Fifo 的数据是在该时钟使能，下一个时钟输出。读使能信号和数据输出有一个时钟的延迟。

主机的读 fifo 使能信号设置为

```
rd_fifo_en =  rd_first_data || (  axi_wvalid && M_AXI_WREADY &&
( !axi_wlast ) );
```

rd_first_data:启动突发传输时的读使能信号，一个时钟有效。

axi_wvalid && M_AXI_WREADY ： 在主机发送有效且从机接受数据时，读取 fifo 中的数据。

!axi_wlast ： 发送最后一个数据时，该数据通过上一个时钟的读使能信号已经读出，在该周期，不再读取新的数据。

4.2.3 数据的读出方案

图像数据储存在 fifo 中，显示驱动模块用于控制图像的显示，通过从 DDR 中读出图像数据，送给显示屏进行显示。数据从 DDR 中读出，送给显示驱动模块时，也要经过 fifo 缓存。

主机先发送读地址（aw_araddr） 以及地址有效信号（ aw_arvalid ）。从机接收到 aw_arvalid 有效信号后，在接收地址信号的同时 ，返回 AW_ARREADY 有效信号，标志地址已经接收完成。

在主机接收到 AXI_ARREADY 有效信号后，便可以开始进行读数据的接收，主机等待从机发送的 AXI_ARVALID 有效信号，此信号有效时，表明读数据总线 AXI_RDATA 上此时有有效的输出数据。此时主机发送 axi_rready 信号，通知从机主机已经开始接收数据，此时从机便可以发送新的数据。

从机在接收到有效数据后，需要将读出的数据在 fifo 中进行缓存。该 fifo 的使能信号 设置为如下：

```
rd_fifo_en = (  axi_rready && AXI_RVALID );
```

AXI_RVALID ： 该信号表示读数据总线上有有效的数据。

axi_rready：该信号有效后，从机在下一个周期才发送新的数据。在该信号无效时，从机发送的数据保持不变。故只有在该信号有效时，将输出数据端口的数据写入到 fifo 中即可。

Fifo 端的写数据，写使能和写数据对齐，数据在下一个时钟写入到 fifo 中去。

4.3 DDR 的数传输控制

4.3.1 总体介绍

DDR 的数据的读写是通过 axi 总线进行数据传输。AXI（Advanced eXtensible Interface)是一种总线协议,该协议是 ARM 公司提出的 AMBA(Advanced Microcontroller Bus Architecture）3.0 协议中最重要的部分，是一种面向高性能、高带宽、低延迟的片内总线。它的地址/控制和数据相位是分离的，支持不对齐的数据传输，同时在突发传输中，只需要首地址，同时分离的读写数据通道。

AXI4 和 AXI4-Lite 包含 5 个不同的通道：

读地址通道；

写地址通道；

读数据通道；

写数据通道；

读响应通道；

用户控制端口分为读控制通道和写控制通道。下面分别做介绍。

4.3.2 写控制通道

用户控制写通道的端口信号如下组成：

信号名	位宽	方向	描述
WR_START	1	input	写开始信号。该信号有效后，启动一次突发传输。
WR_ADRS	32	input	突发传输的起始地址。
WR_LEN	32	input	一次突发传输的数据长度，以字节为单位。
WR_READY	1	output	写状态。当 axi 处于空闲时，该信号为高。其余状态下，该信号为低。
WR_FIFO_RE	1	output	读 fifo（数据）使能信号。该信号有效时，表示需要将有效数据送到写数据端口。
WR_FIFO_EMPTY	1	input	fifo 空标志位。
WR_FIFO_AEMPTY	1	input	fifo 几乎空标志位。
WR_FIFO_DATA	64	input	写数据端口。要传输的数据通过该端口输入。
WR_DONE	1	output	写突发传输完成的标志位。一个时钟高电平。

【注】

1.一般数据是从先缓存在 fifo 中，再从 fifo 中读出，通过 axi 总线再进行一次突发传输。

2.WR_FIFO_RE 是 fifo 的写读能信号，当 axi 总线正在传输数据时，该信号有效，通知数据提供模块发出新的数据。由于数据的输出一般都有一个时钟的延迟，故该信号要在有效数据发送前提前一个时钟有效，保证有效数据正好对齐。

3.传输地址和突发长度是以字节为单位的，当数据位宽为 64bit 时，一次传输的数据为 8 字节，地址增量为 8。一次突发长度为 256 时，突发地址的增量为 256*8 =2048。

4.在进行一次数据突发传输时，将 WR_START 信号置高的同时，将突发首地址和突发长度放置到 WR_ADRS，WR_LEN 端口，然后等待 WR_FIFO_RE 有效时，将要突发传输的数据依次放置于 WR_FIFO_DATA 端口，直到 WR_DONE 信号为高，标志一次突发传输完成。进行下一次的突发传输时，重复以上步骤即可。

5.WR_START 信号可以保持多个时钟有效，但必须在 RD_DONE 信号为高后将该信号置低，否则会自动触发下一次的突发传输。

6，空闲状态下，WR_START 信号要置为低， WR_ADRS，WR_LEN 端口的数据在突发传输开始时更新，在其他时间可保持不变。内部模块会在空闲状态且 WR_START 为高时接收寄存这些数据，启动一次突发传输。

在 WR_READY 信号为高或者接收到 WR_DONE 信号为高后，传输进入空闲状态，此时再启动一次突发传输。或在非空闲态就启动开始信号，将地址和传输长度放置在总线上，等到进入空闲态时，会自动进入下一次突发传输。

4.3.3 读控制通道

用户控制写通道的端口信号如下组成:

信号名	位宽	方向	描述
RD_START	1	input	读开始信号。用于启动一次读突发传输。
RD_ADRS	32	input	读突发首地址。
RD_LEN	32	input	一次突发传输的数据长度，以字节为单位。
RD_READY	1	output	读状态。当 axi 处于空闲时，该信号为高，其余状态下，该信号为低。
RD_FIFO_WE	1	output	写 fifo 使能信号。当该信号有效时，表示读数据端口输出有有效的数据。可用该信号作为

			fifo 的写使能信号，将数据写入到 fifo 中。
RD_FIFO_FULL	1	input	fifo 满标志位。
RD_FIFO_AFULL	1	input	fifo 将满标志位。
RD_FIFO_DATA	64	output	读数据端口。axi 总线读出的数据通过该端口输出。
RD_DONE	1	output	读突发传输完成的标志位。一个时钟高电平。

【注】

1，一般数据读出后先缓存到 fifo 中，等 fifo 中的数据达到一定数量时，再由其他模块读取。

2，RD_FIFO_WE 信号是输出数据的有效信号，当该信号为高时，表示读数据端口有有效的数据输出。可将该信号作为 fifo 的写使能信号。该信号与输出数据保持对齐。

3，传输地址和突发长度是以字节为单位的，当数据位宽为 64bit 时，一次传输的数据为 8 字节，地址增量为 8。一次突发长度为 256 时，突发地址的增量为 256*8 =2048。

4，在进行一次读突发数据传输时，将 RD_START 信号置高的同时，将突发首地址和突发长度放置到 RD_ADRS，RD_LEN 端口，然后等待 RD_FIFO_RE 有效时，有效的读数据从 RD_FIFO_DATA 端口输出，直到 RD_DONE 信号为高，标志一次突发传输完成。进行下一次的突发传输时，重复以上步骤即可。

5，RD_START 信号可以保持多个时钟有效，但必须在 RD_DONE 信号为高后将该信号置低，否则会自动触发下一次的突发传输。

6，空闲状态下，RD_START 信号要置为低， RD_ADRS，RD_LEN 端口的数据在 突发传输开始时更新，在其他时间可保持不变。内部模块会在 空闲状态且 RD_START 为高时接收寄存这些数据，启动一次突发传输。

7，在 RD_READY 信号为高 或者接收到 RD_DONE 信号为高后，传输进入空闲状态，此时再启动一次突发传输。或在非空闲态就启动开始信号，将地址和传输长度放置在总线上，等到进入空闲态时，会自动进入下一次突发传输。

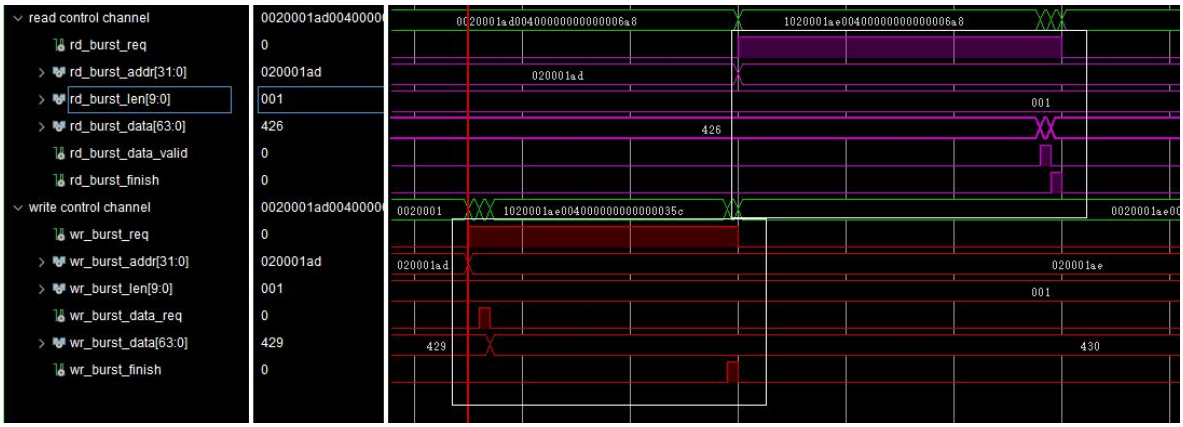
4.3.4 数据传输设计的一些细节

1，突发长度最小可以设置为 1.

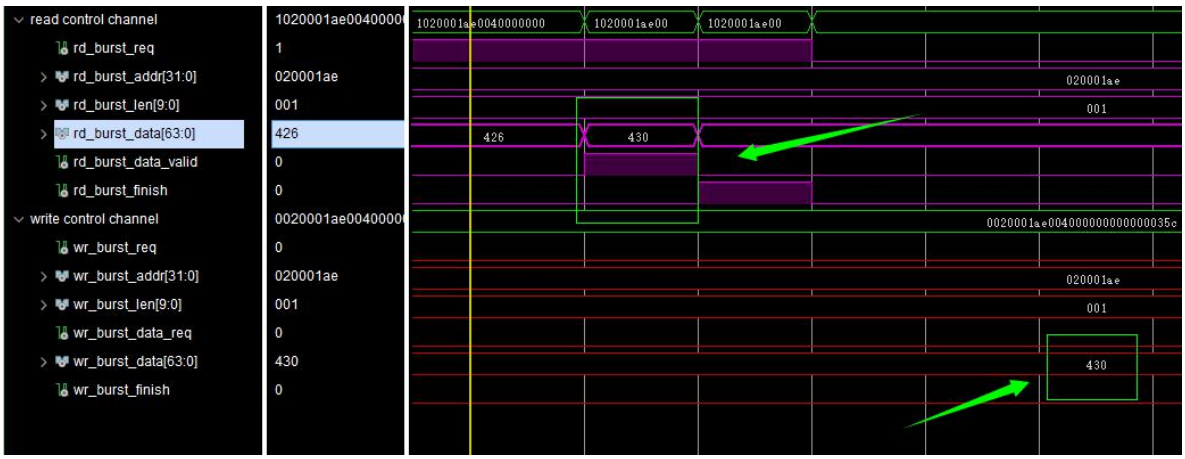
2，RD_START，WR_START 信号在启动一次数据传输时可以一直为高，但要在 RD_DONE，WR_DONE 为高时将该信号置低，否则会自动触发下一次传输，造成数据传输错误。

3.读出的数据和 RD_FIFO_WE 标志位对齐。写入的数据为 WR_FIFO_RE 信号有效之后传输进写数据端口的数据。

如下图，对应的是突发长度为 1 的写+读数据验证，先向设定的地址写入长度为 1 的数据，然后再读出该地址的数据。比较写入的数据和读出的数据是否一致。

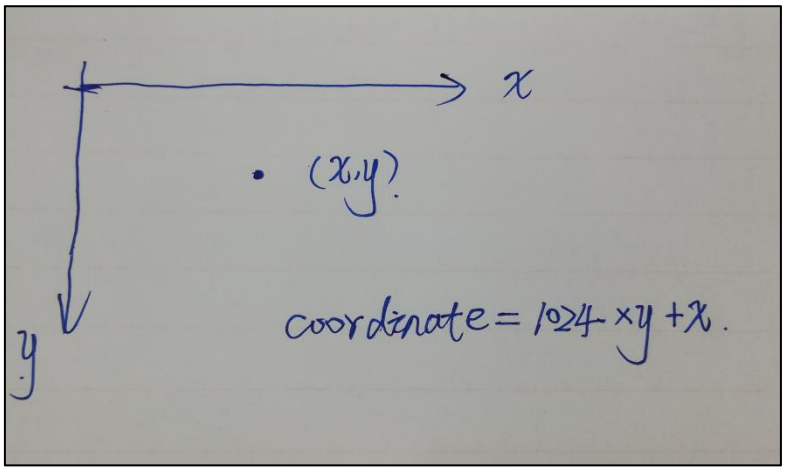


如图，写入的数据为 430，读出的数据也为 430.



4.4 坐标对应关系

每一帧图片突发传输开始时，突发写索引变量从 0 开始记录写入的数据的个数。总线宽度为 64bit,每次写入 8 字节的数据。每个像素点占据 16bit，则一次写入 4 个像素点的数据。故向 ddr 中每写入一个数据，突发写索引变量加 1，图像的相对坐标值加 4。如图所示：



图像的尺寸是 1024*768 。一个像素点占据 16bit（2 字节），故一张图片占据的储存空间为 ：1024*760*2 = 1572864(10) =（180000）16。共要向 ddr 中写入 1024*768/4 = 196608 次= （30000）16 数据，才能将一帧数据储存到 ddr 中。

从左到右记为屏幕的 X 坐标，从上到下记为屏幕的 Y 坐标，图像的宽度（X 方向）为 1024 。X 从左到右，从 0 开始依次增加，Y 从上到下，从 0 开始依次增加。

则屏幕的相对坐标为 1024 * y + x 。 突发写索引加 1，图像的相对坐标加 4 。突发写索引表示为： write_addr_index。屏幕坐标表示为：coordinate_screen 。有 coordinate_screen =1024 * y + x 。 两者的对应关系为：write_addr_index = coordinate_screen[MSB : 2] 。

即要控制屏幕坐标为 coordinate_screen(1024 * y + x)的点的显示颜色，只需要改变突发写索引表示为 coordinate_screen[MSB : 2] 对应的写入数据即可。而 coordinate_screen[1:0]控制改变四个像素点中的哪一个。

要读取坐标为 coordinate_screen(1024 * y + x)的颜色值，只需要将地址索引设置为 coordinate_screen[MSB : 2] ，将对应储存在 ddr 中的数据读出即可。coordinate_screen[1:0]用于选择读出的 4 个像素点中的哪一个。

4.5 ddr+串口联合测试

4.5.1 总体设计

DDR 端的数据通过 AXI 总线进行数据传输。在前面章节介绍了 DDR 数据读写模块的设计（aq_axi_master），本章节中便对这个 axi 的读写模块进行测试。在测试中，先向 ddr 的某个地址中写入数据，然后再将该地址的数据读取出来，通过串口将此数据发送到电脑端，以此验证 ddr 数据的读写是否正确。

4.5.2 串口收发模块设计

该串口收发模块有串口发送模块，串口接收模块，波特率生成模块，发送数据 fifo

模块，接收数据的 fifo 模块组成。

默认配置下，要求输入的参考时钟为 50MHz，输入输出的波特率默认配置为 115200. 该设置体现在波特率生成模块中。

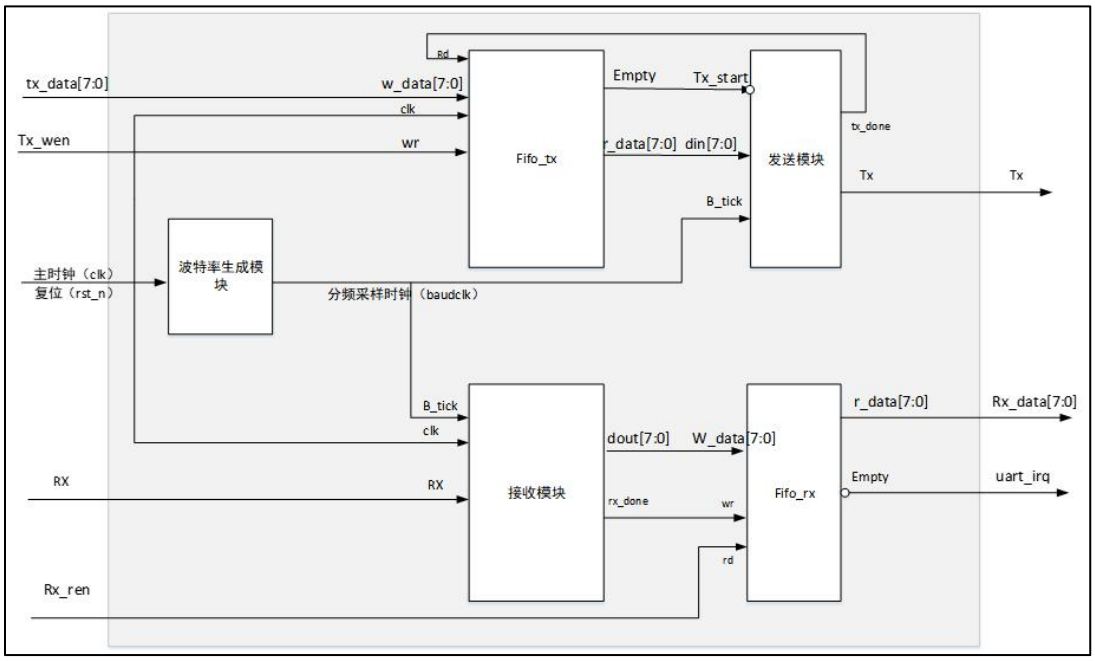
波特率可以通过定制化参数来修改。

```
parameter    SYS_CLK_FRP      =    50_000_000,
parameter    BAUDRATE         =    115200
```

SYS_CLK_FRP 为串口模块的输入时钟频率。

BAUDRATE 为要设置的数据传输波特率。

整体的设计框架如下：



端口定义如下：

端口名	方向	描述
Clk	Input	主时钟，默认频率为 50MHz

Rst_n	Input	复位信号，低有效
uart_rx	Input	串口接收端口
uart_tx	Output	串口发送端口
tx_data[7:0]	Input	tx 端要发送的 1 字节数据
rx_data[7:0]	output	rx 端口接收到的 1 字节数据
tx_en	Input	发送数据的写入使能信号
rx_done	output	串口接收到数据的有效标志位

该模块的使用方法：

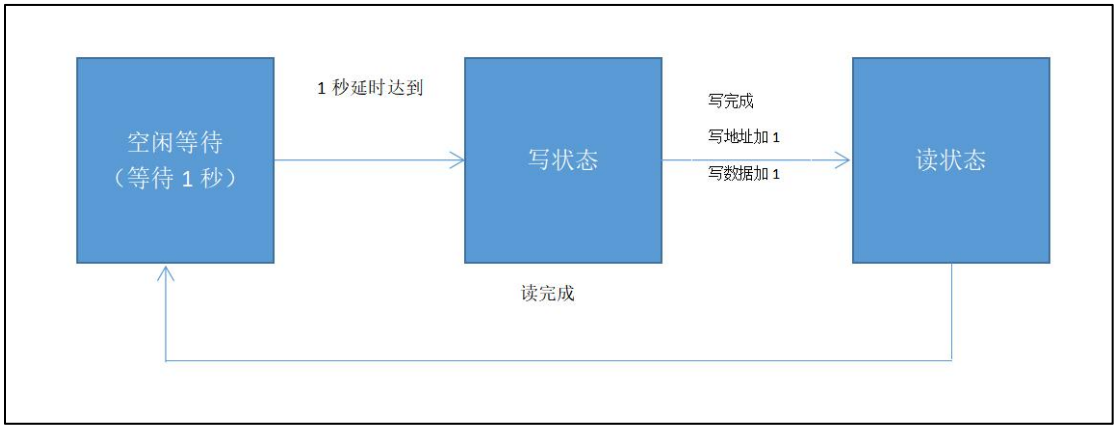
发送数据：

将要发送的数据放置到 Tx_data_in[7:0]总线上，同时将 Tx_wen 置为高，下一个时钟时，该数据会写进 tx_fifo 中。如要发送多字节数据，则重复上述步骤，每个时钟向 TX_FIFO 中发送一个字节的数据，tx_fiof 的默认深度为 256，可以储存 256 个字节的数据，则一次最多可以发送 256 字节数据。

接收数据：

接收到一个字节的有效数据时，接收数据标志位 rx_done 为高，同时在 rx_data 端口输出接收到的数据

4.5.3ddr 端的数据读写设计



ddr 端的读写设计入上图所示。首先进入空闲态，延时等待 1 秒，计时达到 1 秒后，进入写数据状态。起始状态下写地址和写数据设置为 0，向该地址写入数据后，写地址和写数据均加 1，然后读取此次写入地址的数据，待读完成后，进入空闲态，再次延时等待 1 秒，计时达到 1 秒后，在进行下一次的数据读写验证。

```
else
begin
    case(state)
        IDLE:
        if( time_cnt == 50_000_000   )
        begin
            state               <=  MEM_WRITE;
            wr_burst_req        <=  1'b1;

        end
        MEM_WRITE:
        begin
            if(wr_burst_finish)
            begin
                state               <=  MEM_READ;

                wr_burst_req        <=  1'b0;

                rd_burst_req        <= 1'b1;
                rd_burst_addr       <=  wr_burst_addr;

            end
        end
        MEM_READ:
        begin
            if(rd_burst_finish)
            begin
                state               <=  IDLE;

                rd_burst_req        <= 1'b0;

                write_read_len      <=  write_read_len + BURST_LEN;
                wr_burst_addr       <=  wr_burst_addr + BURST_LEN;
                wr_burst_data_reg   <=    wr_burst_data_reg +   1'b1;

            end
        end
```

空闲延时等待

数据写入

数据读取

4.5.4 串口发送数据设计

从 ddr 中读出的数据是 64bit，但串口一次只能发送 8bit 的数据，则要将这 64bit 的数据分为 8 次发送出去。设计了一个数据发送模块，将 64bit 的数据转换为 8 个 8bit

的数据，依次通过串口发送。

具体的实现方法是，设置一个计数器，待收到 ddr 读出的有效数据后，将该数据份 8 次传输到串口发送模块。同时将串口发送数据的有效标志位置高，直到 8bit 的数据全部发送完成。

核心代码如下图所示：

```
always@(posedge clk,negedge rst_n )
begin
    if(!rst_n)
        o_en    <=  1'b0;
    else if( cnt == 7 )
        o_en    <=  1'b0;
    else if( i_en )
        o_en    <=  1'b1;
    else
        o_en    <=  o_en;
end
```

发送数据有效标志位

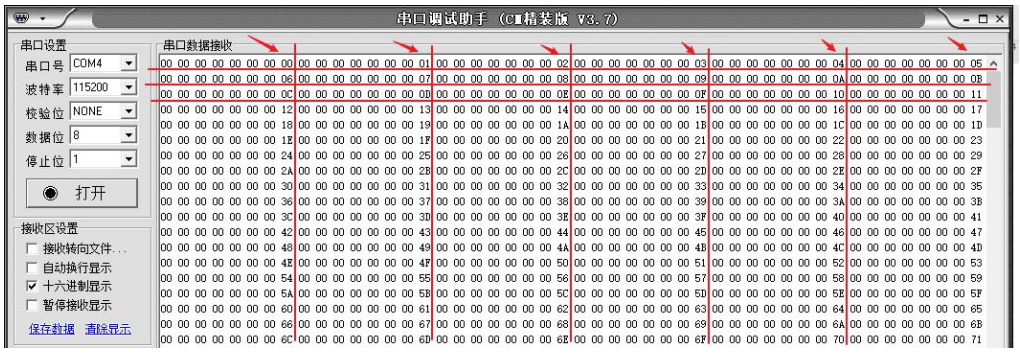
```
always@(*)
begin
    case(cnt)
        0       :   o_data  =   i_data_reg[63:56];
        1       :   o_data  =   i_data_reg[55:48];
        2       :   o_data  =   i_data_reg[47:40];
        3       :   o_data  =   i_data_reg[39:32];
        4       :   o_data  =   i_data_reg[31:24];
        5       :   o_data  =   i_data_reg[23:16];
        6       :   o_data  =   i_data_reg[15:8];
        7       :   o_data  =   i_data_reg[7:0];
        default :   o_data  =   8'b0;
    endcase

end
```

数据的串行转换

4.5.5 测试结果

将经过综合，布局布线后，生成的比特流文件下载到开发板中，打开串口终端，连接到 FPGA 的串口。可以看到数据以 1 秒 8 字节的速度在不断的接收。

而且接收到的数据以 8 字节为一组，以依次加一的规律出现。证明 ddr 的数据读写过程无误。



4.6 图像旋转方案

4.6.1 总体方案

标准模式下，从摄像头获取到图像数据，将该图像数据缓存到 DDR 中，再通过显示驱动模块将图像读取出来，在显示屏上进行显示。

图像数据通过 AXI 接口写入到 DDR 中，通过 AXI 总线从 DDR 中读取。这期间要跨三个时钟域。分别是 摄像头数据输出时钟，AXI 读写时钟，显示屏驱动时钟。在跨时钟域传输数据时，数据都要经过 fifo 缓存。

在图像旋转设计中，插入一个图像旋转模块。将从摄像头缓存的图像先读取出来，组合成一帧旋转的图像后再写入 ddr 中，再由显示驱动模块读取进行显示。

4.6.2 数据传输方案

ddr 中数据的读取采用 AXI 协议。数据从摄像头写入 ddr，从 ddr 读出传输到显示模块，均采用 axi 的突发传输。因为数据都是按照相同的顺序进行储存和读取，故只需要按照顺序进行数据的突发写入和读取即可正确的显示一张图片。

而在进行旋转操作中，由于旋转后的图片和原图的坐标不是顺序对应的，旋转输出图像数据由若干个不是顺序排列的原图像数据决定的，故对于原图像数据的读取，利用突发传输反而浪费时间，且突发读取到的数据中可用的数据占比较少。

对原图像的数据读取拟采用突发长度为 1 的传输。根据旋转图像的所需要的原始图像的数据来读取所需地址的数据，用于重建旋转后的图像。

旋转后的图像数据也经过突发长度为 1 的方式写入进 ddr 中。

旋转图像的重建模块的始终频率设置为于 axi 的时钟频率一致，一来可以不使用 fifo 来数据缓存，二来，该时钟频率为 100MHz，运行速度也更快。

4.6.3 图像帧处理

在读取原图时，如果原图像在不停地储存更新，那么重建的旋转是由多帧图像组合而成的，该图像便会出错。如果在旋转图像储存过程中便读取该图像进行显示，显示图像的帧率大于旋转图像重建的帧率，显示的图像也会出错。

该方案采用了降帧的方案。在图像储存时，不对输入的每一帧图像都进行储存。当储存完了一张图后，停止储存下一帧的图片，然后旋转控制模块便开始读取这一帧图片，进行旋转重建，待到这一帧图片旋转重建完成后，才开始接受下一帧的图片。这样便保障了读取时原图的完整性。

在将图片重建后，需要进行储存，利用乒乓操作，将重建的图像利用两个空间进行储存。当向空间 1 写入重建的图像数据时，不断读取空间 2 的图像数据进行显示。直到空间 1 的一帧图像数据写入完成，且该帧显示结束，交换读写地址，将重建的图像数据

写入到空间 2，同时读取空间 1 的数据进行显示。由于重建的帧率小于显示的帧率，一个空间的图像数据需要重复显示多次。

如上所述，在该方案中，原图的输入，旋转图像的重建都进行了降帧处理。但图像显示没有做降帧处理，但在没有交换读写地址时，会重复显示储存在该空间的一帧图片，呈现出动态刷新，静态显示的效果。

4.7 图像旋转计算

4.7.1 图像旋转原理

图像旋转的本质利用的是向量的旋转，而在 MATLAB 等算法工具中向量的计算往往转换成相应矩阵的计算，向量是几何中的概念，因此在算法的编译中常常不直接进行向量的运算，而是将其转换成在极坐标中的对应坐标矩阵来进行算法的构建。

矩阵乘法的实质是进行线性变换，因此对一个向量进行旋转操作也可以，通过矩阵和向量所对应的特征矩阵相乘的方式进行，而这在大多数的计算机语言中是通用的方法。正是因为这一点，在图像旋转的这个模块中，采用了构建特征矩阵进行坐标转化这个思路。

具体思路如下。假设有二维向量 v = [x ; y]，其中 x,y 是原图的像素点的横轴和纵轴坐标。若要进行逆时针旋转角度 a。则旋转矩阵 R 为：

$$\begin{bmatrix} \cos(a) , & -\sin(a) \\ \sin(a) , & \cos(a) \end{bmatrix}$$

旋转后的向量 Ro = R * v。 Ro =[Xo,Yo]; 其中 Xo, Yo 是输出图像的坐标值。

在正式处理过程中可以这么表示，原像素位置记为 p，中心点记为 c，旋转后像素位置记为 pp。则有(pp - c) = R*(p - c)，即:

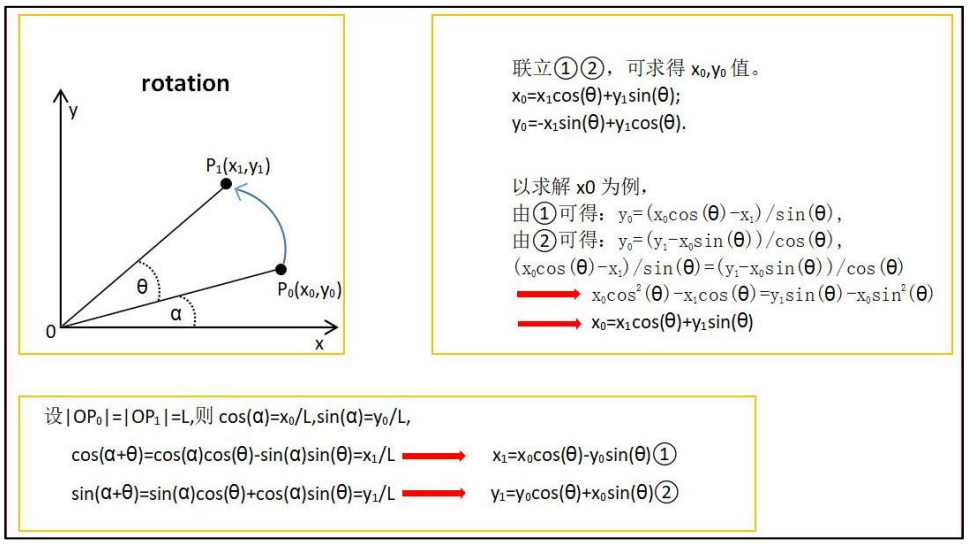
$$pp = R*(p-c) + c$$

4.7.2 输入输出图像坐标的方案选择

方案一:

在此方案中，实现代码的方式是正向的思路，将原图中的像素点的坐标进行坐标的旋转，然后直接幅值到输出的图像中，此方案旨在找到输入坐标与输出坐标之间的代数对应关系，以此来进行 Verilog 代码的编写。

但在实际的分析的过程中，先采用极坐标系进行分析，得到了对应的坐标对应关系，如下图所示：



在该方法中，首先将原始坐标以及目标坐标放入了极坐标中，并且通过在极坐标中的关系，找到了同时满足 X0,Y0,X1,Y1 四个参量的方程组，以此来解出对应的坐标关系，并以此为基础得到了输入与输出之间的矩阵运算关系如下：

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这样就解决了坐标的代数关系。但在实际的测试中发现，这种方法所旋转得到的图像有着较为严重的失真现象，具体情况如下图所示：

很明显可以看到，在旋转之后这两张图片出现了较大的差别，首先是原图像被裁减了，其次是目标图像中有较多的瑕点(杂点)。究其原因在于，从原图旋转后得到的目标图像的像素位置在原图中找不到。另外就是边缘被裁剪的问题，由于在这个方案中约束了显示区域，因此在旋转的过程中，部分像素点就会由于超出边界而被裁剪。针对以上的两个问题，进行了如下改进。

方案二:

由于在之前的方案中出现了杂点以及图像边缘裁剪的问题，因此在本方案中，我们采用了逆向思维，用目标图像的坐标去与原图的坐标进行坐标匹配，若在原图像中能找到匹配的图像，就显示该点旋转后的点坐标，若在原图中找不到该点，则不显示该点，通过这样就解决了杂点的问题，具体所限定的原图的查找区域代码如图所示：

```
pp = round(R*(p-c)+c);
%逆向进行像素的查找
if (pp(1) >= 1 && pp(1) <= h && pp(2) >= 1 && pp(2) <= w)
    im2(i, j, k) = im(pp(1), pp(2), k);
end
```

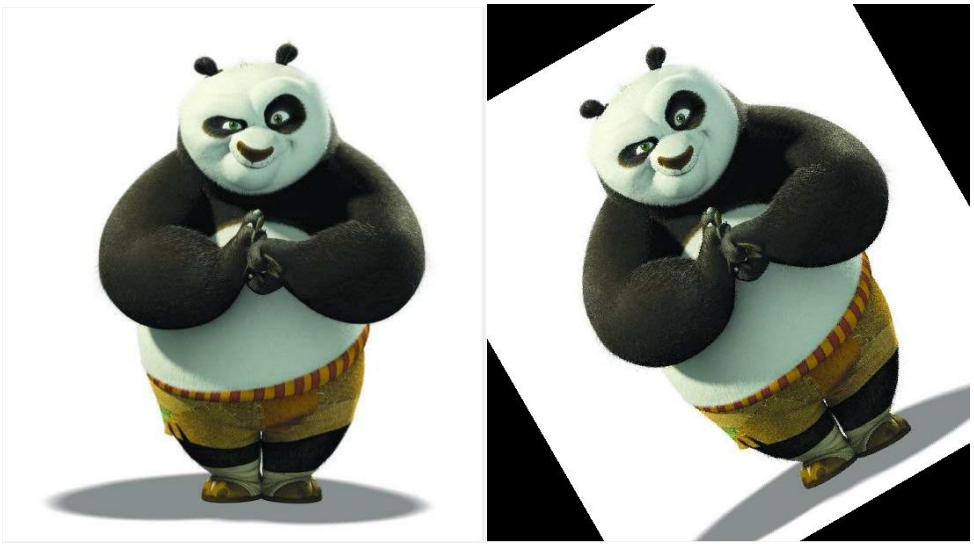
其中，pp 为旋转在后的坐标对应矩阵，在 if 语句中限定了原图的区域，用此区

域则可以到原图中的坐标点，以此来排除不在区域中的坐标点，这样就可以解决杂点的问题。(以上是前期在 MATLAB 中的仿真代码截取)

在这种方案下，坐标的对应关系如下：

$$\begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这样，该旋转后的图像就有了较好的还原度，达到了相应的题目要求，具体的方案的效果如下图所示：



如图所示，相对方案一而言，图像的效果就好了很多，但图像边缘仍然存在边缘被切割的现象，因此在第三种方案中，对代码进一步进行优化。

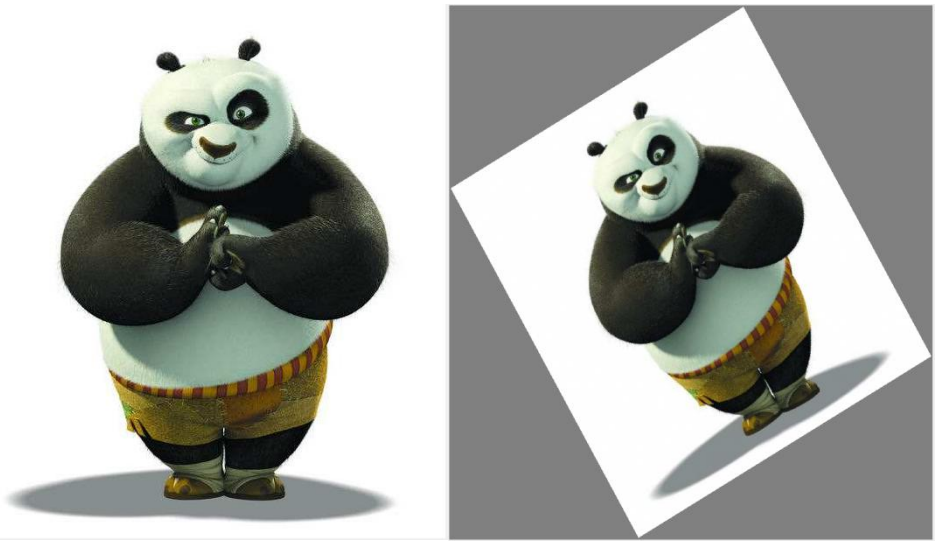
方案三:

考虑到未对旋转后的图像进行显示区域的划分，因此此类旋转只是对单一像素点的旋转，然后在原图像的显示区域上进行坐标点的重新组合，得到显示的图像。在解决的方法的思路上，采用目标显示区域的重新划分来解决该问题。

具体思路是，采用原图像的长宽作为基准，再用坐标转换的关系，将长和宽转换到旋转后的坐标系中，得到目标图像在旋转后坐标系中的显示区域，具体如下：

```
% 计算显示完整图像需要的画布大小
hh = floor(w*sin(a)+h*cos(a))+1;
ww = floor(w*cos(a)+h*sin(a))+1;
c2 = [hh; ww] / 2;
```

这样，就解决了图像边缘被裁剪的问题，是整个图像得以完整的显示，实际的效果如下：



从图示的效果可以看出，边缘区域被裁剪的问题被解决了，但问题是图片加阴影的区域面积比原图大很多。

但在实际的操作中，采用这一类的图像点坐标的对应关系，产生的结果与预期有着较大的误差，图像的效果较差，因此为了更好的进行图像的处理，我们又在网络上寻找了 CORDIC 算法，以此来得到更好的处理效果。

方案四：

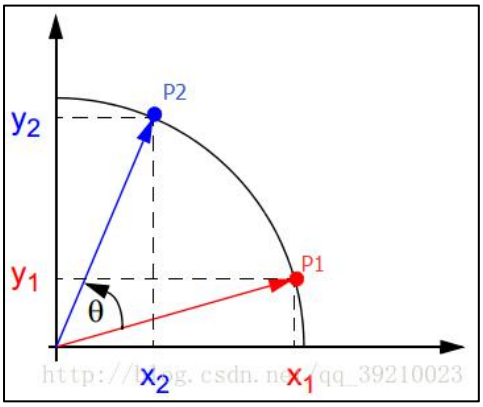
坐标旋转数字计算机 CORDIC(COordinate Rotation DIgital Computer)算法，通过移位和加减运算，能递归计算常用函数值，如 Sin，Cos，Sinh，Cosh 等函数，由 J. Volder 于 1959 年提出，首先用于导航系统，使得矢量的旋转和定向运算不需要做查三角函数表、乘法、开方及反三角函数等复杂运算。J. Walther 在 1974 年用它研究了一种能计算出多种超越函数的统一算法。

该算法在计算机语言中广泛的应用，使复杂的问题变为计算机易于操作的算式，大大的提高了计算机的优势。CORDIC 算法是一个“化繁为简”的算法，将许多复杂的运算转化为一种“仅需要移位和加法”的迭代操作。CORDIC 算法有旋转和向量两个模式，分别可以在圆坐标系、线性坐标系和双曲线坐标系使用，从而可以演算出 8 种运算，而结合这 8 种运算也可以衍生出其他许多运算。

	旋转模式	向量模式
圆坐标系	cos & sin	$\tan^{-1}$
线性坐标系	*	/
双曲线坐标系	cosh & sinh	$\tanh^{-1}$

首先，我们先了解 CORDIC 算法的几何原理：

假设在 xy 坐标系中有一个点 P1（x1，y1），将 P1 点绕原点旋转 θ 角后得到点 P2（x2，y2）。



于是可以得到 P1 和 P2 的关系。

$$x2 = x1\cos\theta - y1\sin\theta = \cos\theta(x1 - y1\tan\theta)$$

$$y2 = y1\cos\theta + x1\sin\theta = \cos\theta(y1 + x1\tan\theta)$$

以上就是 CORDIC 的几何原理部分，从原理中，我们可以知道，当已知一个点 P1 的坐标，并已知该点 P1 旋转的角度 θ，则可以根据上述公式求得目标点 P2 的坐标。然后，

麻烦来了，我们需要用 FPGA 去执行上述运算操作，而 FPGA 的 Verilog 语言根本不支持三角函数运算。因此，我们需要对上述式子进行简化操作，将复杂的运算操作转换为一种单一的“迭代位移”算法。

CORDIC 的优化算法介绍

我们先介绍算法的优化原理：将旋转角 θ 细化成若干分固定大小的角度 θ i，并且规定 θ i 满足 tan θ i = 2-i，因此∑ θ i 的值在[-99.7°，99.7°]范围内，如果旋转角 θ 超出此范围，则运用简单的三角运算操作即可（加减 π）。

然后我们需要修改几何原理部分的假设，假设在 xy 坐标系中有一个点 P0（x0，y0），将 P0 点绕原点旋转 θ 角后得到点 Pn（xn，yn）。

于是可以得到 P0 和 Pn 的关系。

$$xn = x0\cos\theta - y0\sin\theta = \cos\theta(x0 - y0\tan\theta)$$

$$yn = y0\cos\theta + x0\sin\theta = \cos\theta(y0 + x0\tan\theta)$$

然后，我们将旋转角 θ 细化成 θ_i，由于每次的旋转角度 θ_i 是固定不变的（因为满足 $\tan\theta i = 2^{-i}$），如果一直朝着一个方向旋转则∑ θ_i 一定会超过 θ（如果 θ 在[-99.7°，99.7°]范围内）。因此我们需要对 θ_i 设定一个方向值 di。如果旋转角已经大于 θ，则 di 为-1，表示下次旋转为顺时针，即向 θ 靠近；如果旋转角已经小于 θ，则 di 为 1，表示下次旋转为逆时针，即也向 θ 靠近。然后我们可以得到每次旋转的角度值 di θ_i，设角度剩余值为 zi+1，则有 zi+1 = zi - di θ i，其中 z0 为 θ。如此随着 i 的增大，角度剩余值 zi+1 将会趋近于 0，此时运算结束。（注：可以发现，di 与 zi 的符号位相同）

第一次旋转 θ 0，d0 为旋转方向：

$$x1 = \cos\theta 0(x0 - d0y0\tan\theta 0)$$

$$y1 = \cos\theta 0(y0 + d0x0\tan\theta 0)$$

由于 i 从 0 至 n-1，所以上式可以转化成下式：

$$xn = 1/\prod\cos\theta_i(x0\cos\theta - y0\sin\theta)，（其中 i 从 0 至 n-1）$$

$$yn = 1/\prod\cos\theta_i(y0\cos\theta + x0\sin\theta)，（其中 i 从 0 至 n-1）$$

注意：上式中的 xn，yn 是经过迭代后的结果，而不是之前假设中的点 Pn（xn，yn）。了解这点是十分重要的。

根据高中学的三角函数关系，可以知道 $\cos\theta i = 1/[(1+\tan2\theta i)\hat{}0.5] = 1/[(1+2-2i)\hat{}0.5]$，而 $1/[(1+2-2i)\hat{}0.5]$的极值为 1，因此我们可以得出一个结论：当 i 的次数很大时，$\prod\cos\theta i$ 的值趋于一个常数。

因此，我们就成功的得到了一个适合 FPGA 执行的旋转算法，使得图像的质量大为提升。用 matlab 产生的查找表如下所示：

i	θ_i（arctan（2^{-i}），单位°）	$\cos\theta_i$	$\prod\cos\theta_i$	$1/\prod\cos\theta_i$
0	45.0	0.7071067812	0.7071067812	1.414213562
1	26.56505118	0.894427191	0.632455532	1.58113883
2	14.03624347	0.9701425001	0.6135719911	1.629800601
3	7.125016349	0.9922778767	0.6088339125	1.642484066
4	3.576334375	0.9980525785	0.6076482563	1.645688916
5	1.789910608	0.9995120761	0.6073517701	1.646492279
6	0.8951737102	0.999877952	0.6072776441	1.646693254
7	0.4476141709	0.9999694838	0.6072591123	1.646743507
8	0.2238105004	0.9999923707	0.6072544793	1.64675607
9	0.1119056771	0.9999980927	0.6072533211	1.646759211
10	0.05595289189	0.9999995232	0.6072530315	1.646759996
11	0.02797645262	0.9999998808	0.6072529591	1.646760193
12	0.01398822714	0.9999999702	0.607252941	1.646760242
13	0.006994113675	0.9999999925	0.6072529365	1.646760254
14	0.003497056851	0.9999999981	0.6072529354	1.646760257
15	0.001748528427	0.9999999995	0.6072529351	1.646760258

（以上部分过程摘取自 CSDN）

综合以上四种方案，结合实际需求，由于我们的显示是在一块固定大小的屏幕上进行显示，整个图像的显示范围有限，采用 CORDIC 算法进行坐标变换产生的延时太大。最终基于处理速度和资源占用的均衡考虑，最终选择方案二作为我们图像旋转的设计方案。

4.7.3 旋转坐标计算

在该设计中，要求图像拥有 0 到 360 的任意角度的旋转，坐标变换需要角度的正弦和余弦值。

利用 matlab 生成正余弦表，并将其扩大 256 倍，打印到文件中。利用得到的正余弦表数值，将其写入 verilog 代码中，生成正余弦查找表。通过输入角度值来索引其正余弦数值。Matlab 生成正余弦列表的代码如下；

```
x = 0:pi/180 :pi*2;

y1=fix ( 2^8 *sin(x) );
y2=fix ( 2^8 *cos(x) );

fid=fopen('sin.txt','w');
fid2=fopen('cos.txt','w');

 fprintf(fid,"角度x（单位：度）\t:\t 正弦值（sin（x））\n");
 fprintf(fid2,"角度x（单位：度）\t:\t 余弦值（sin（x））\n");

for i = 1:360
    fprintf(fid,"%d\t:\t %d\n",i-1,y1(i));
    fprintf(fid2,"%d\t:\t %d\n",i-1,y2(i));
end
```

该正弦，余弦通过 MATLAB 计算得到，并预先储存到 FPGA 的片上储存空间中，在进行坐标变换时，读取对应角度的正弦，余弦值，进行坐标变换。由于计算得到的正弦和余弦值为浮点数，而 FPGA 擅长于进行整数运算。故要进行浮点数到整数的转换，具体

的实现方法是，将计算得到的浮点正弦，余弦值乘上 256 后再取整，计算得到的结果于原结果相比被扩大了 256 倍，而在数字电路中，除法操作可以用移位来进行。结果右移 8 位即等效于除于 256 。

坐标变换的核心代码如下：

```
assign  x_rotate_temp   =    ( x_wire <<< 8 ) * cos_value   -   ( y_wire <<< 8 ) * sin_value;
assign  y_rotate_temp   =    ( x_wire <<< 8 ) * sin_value   +   ( y_wire <<< 8 ) * cos_value;

assign  x_rotate        =   x_rotate_temp   >>> 16;
assign  y_rotate        =   y_rotate_temp   >>> 16;
```

将坐标变换计算模块封装为一个子模块，输入输出图像的坐标和旋转角度后，即可计算出对应的输入图像对应的像素的坐标。然后读取该坐标的像素值，写入到旋转重建的图像对应的坐标位置即可。

4.8 Python 编写上位机

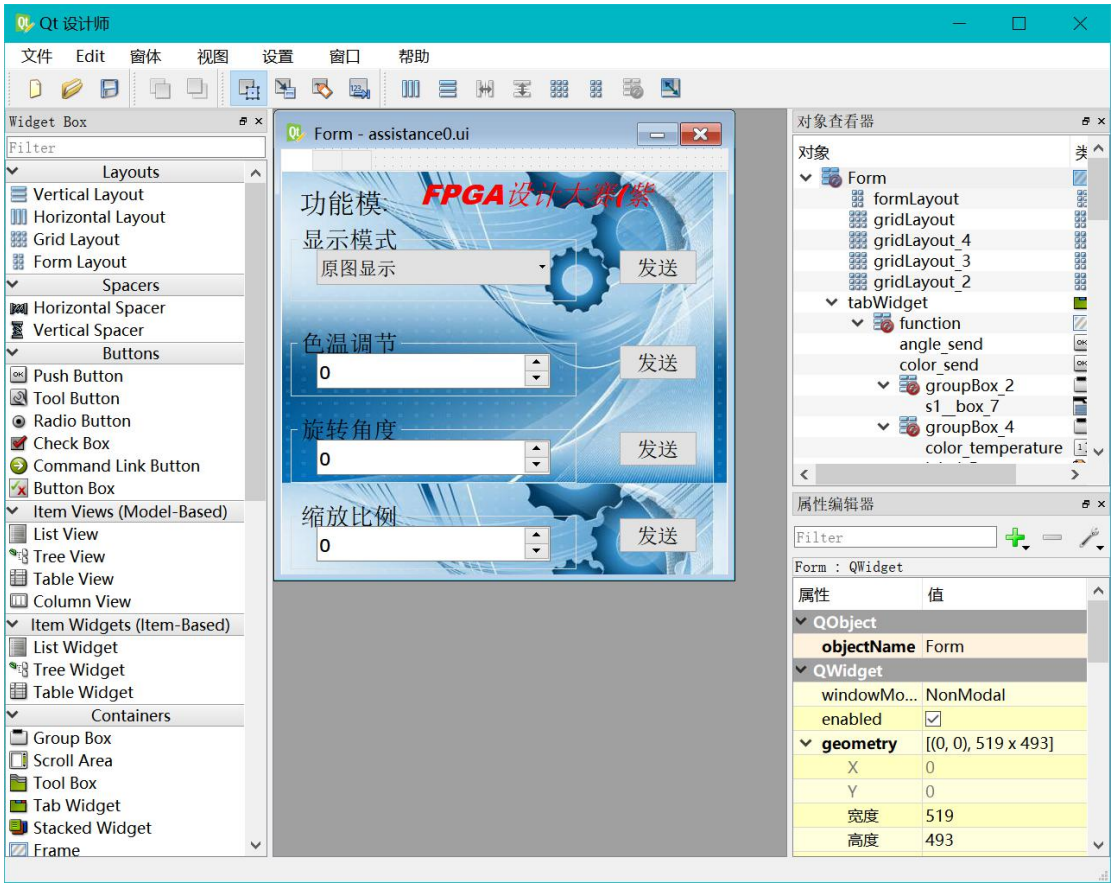
4.8.1 总体方案

采用 python 脚本语言进行上位机的设计，以辅助完成相应的功能控制，通过在上位机的操作，将指令通过串口传递到 FPGA 上已完成对功能的精确的控制。采用 python 中的 PyQt5 模块进行界面 UI 的设计，通过 QT designer 设计界面，让整个 UI 得以完善，实现较好的人机交互。

4.8.2 QT designer 设计方案

Designer 的设计是 PyQt5 的一个交互软件，负责将相应的控件一模块的形式放到 QWidgets 中，实现控件 button，combox,以及 pushbutton 等的组合，以实现较好的界面形式。显示部分分为发送区，接收区以及状态显示区，控制部分分为各种按钮控制以

及数字的控制，实现阶梯数据的改变，发送按钮与串口相连，实现指令的发送。四种功能通过不同的模块进行控制，是它们有机的结合起来，达到控制 FPGA 的整体效果。



4.8.3 串口设计方案

通过 pyserial 的函数调用实现对串口的检测并且使用，以十六进制的数据形式，对指令执行传输 FPGA 的串口进行数据的接收，没接收到一部分数据就进行一次数据的回显，到上位机上，实测延时可以忽略，达到指令的时发时收，是对串口小助手进行的一次较大的提升。设计了 ASCI 和十六进制两种方式进行数据的传输，方便选择。

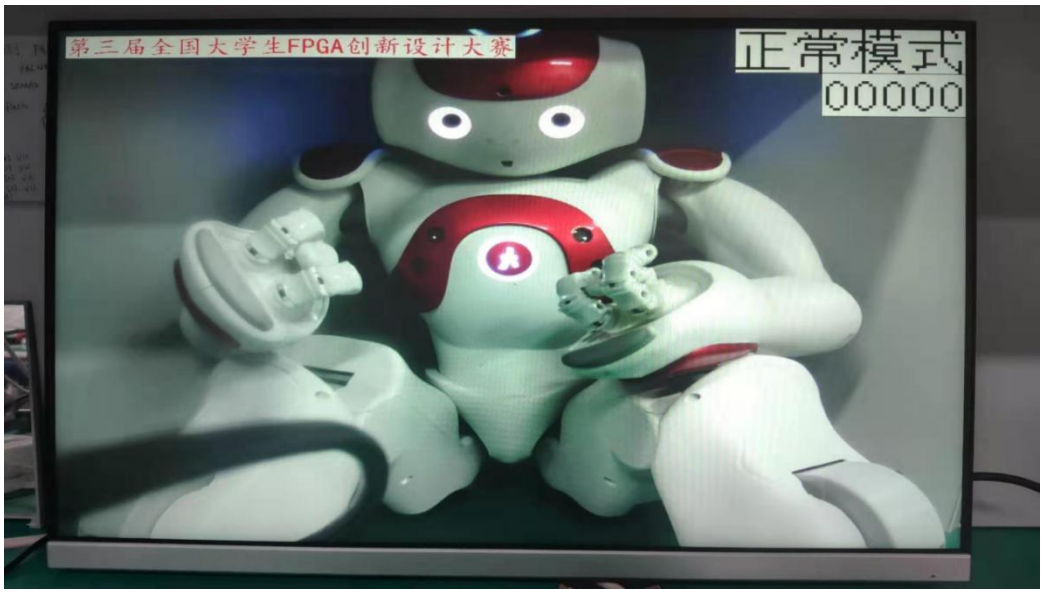
4.8.4 界面的设计方案

在 designer 中设计好之后，采用 PyQt5 中的 Pyuic 的模块进行格式的转换，从 UI 文件转换成 PY 文件，是整个界面转化成代码进行相应的修改，添加和修改控件，并对功能的函数进行编辑优化。Def 函数的使用只一个关键的点，整个界面的函数均由 def 函数定义，实现相应的功能和串口的通信。最后完成的上位机的制作。（主程序调用的程序如下：）

```
#主循环，开始运行程序
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    myshow = Pyqt5_Serial()
    myshow.show()
    sys.exit(app.exec_())
```

第五部分 效果展示(Results show)

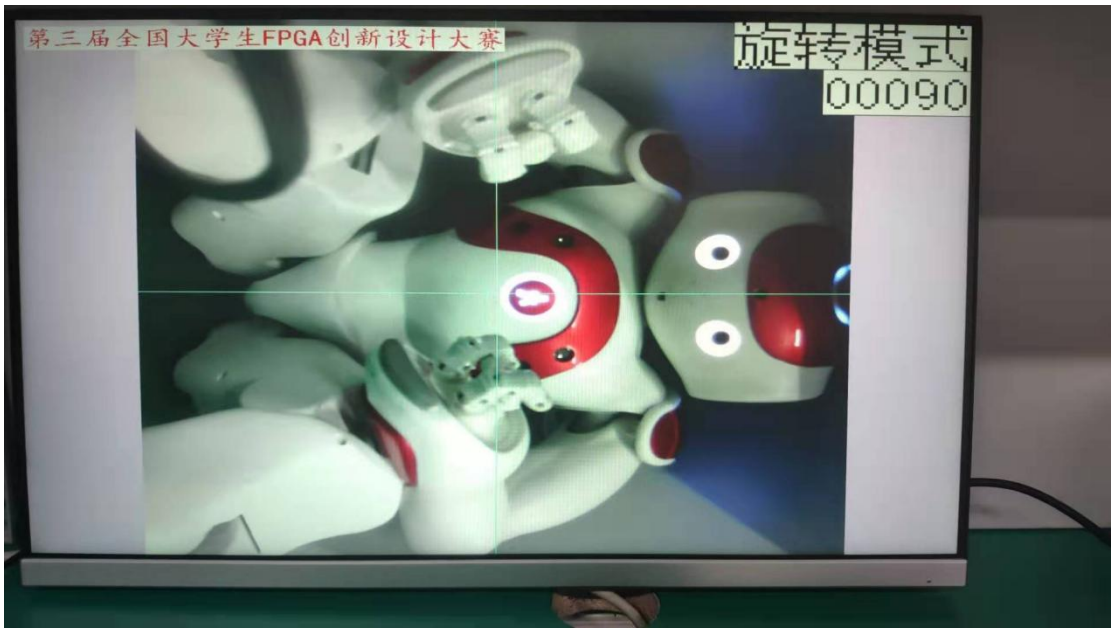
5.1 原图 (正常显示)



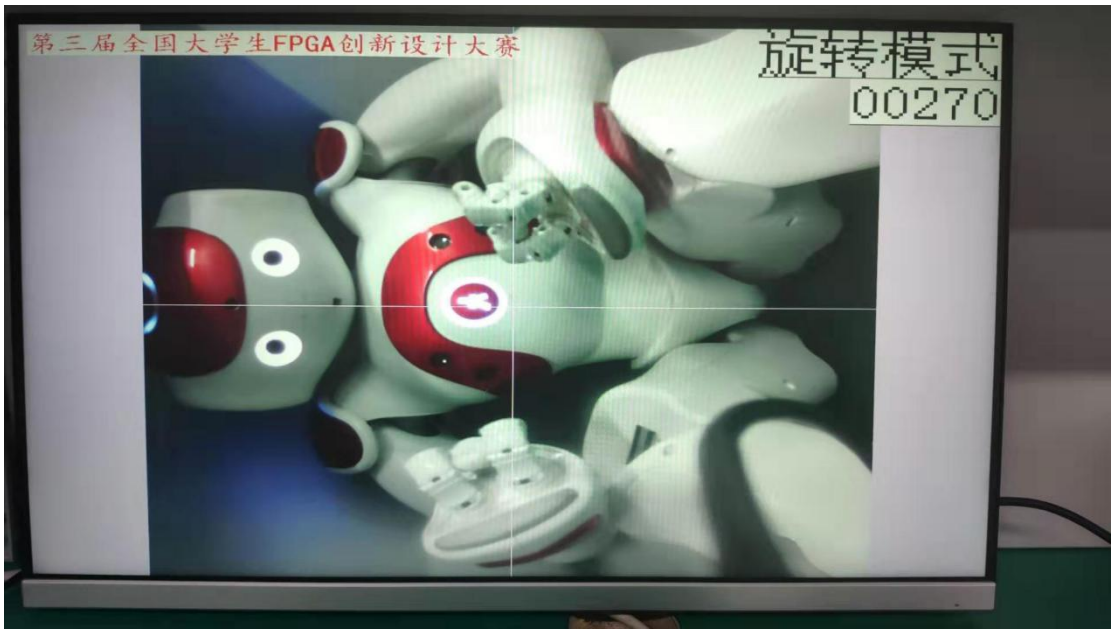
5.2 旋转模式

5.2.1,顺时针旋转 22 度

5.2.2,顺时针旋转 90 度



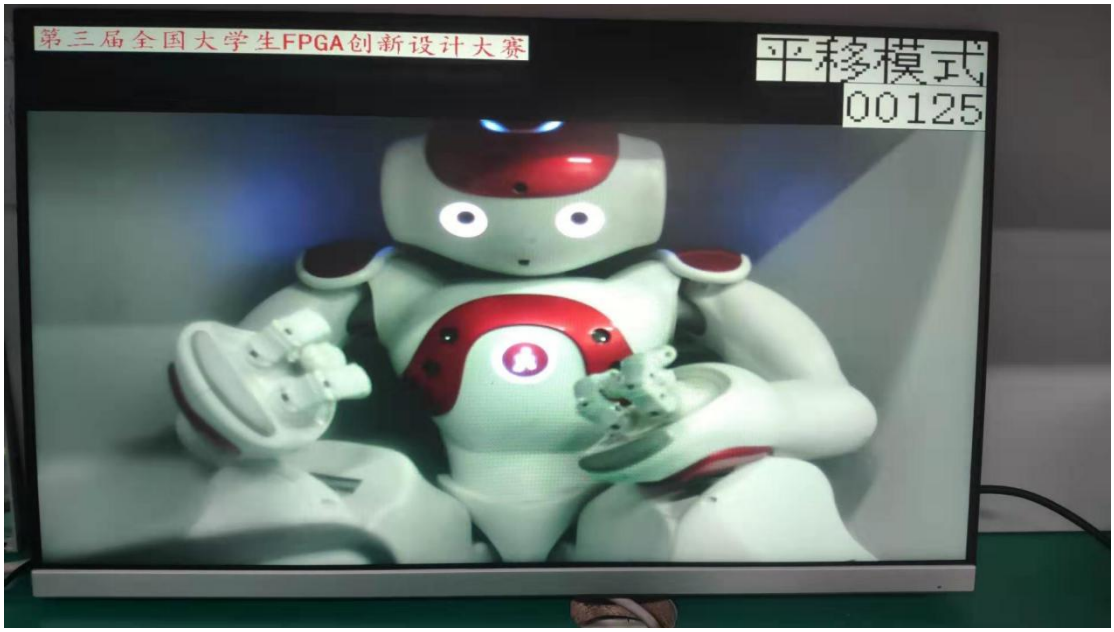
5.2.3,顺时针旋转 270 度



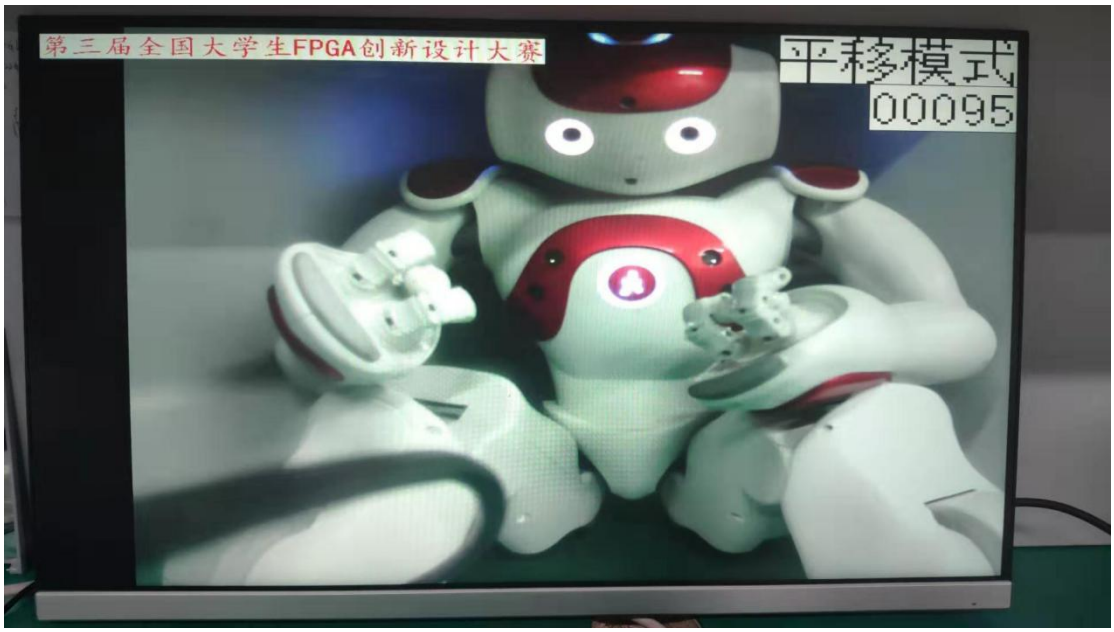
5.2.4,顺时针旋转 341 度

5.3 平移模式

5.3.1,向下平移 125 个像素点

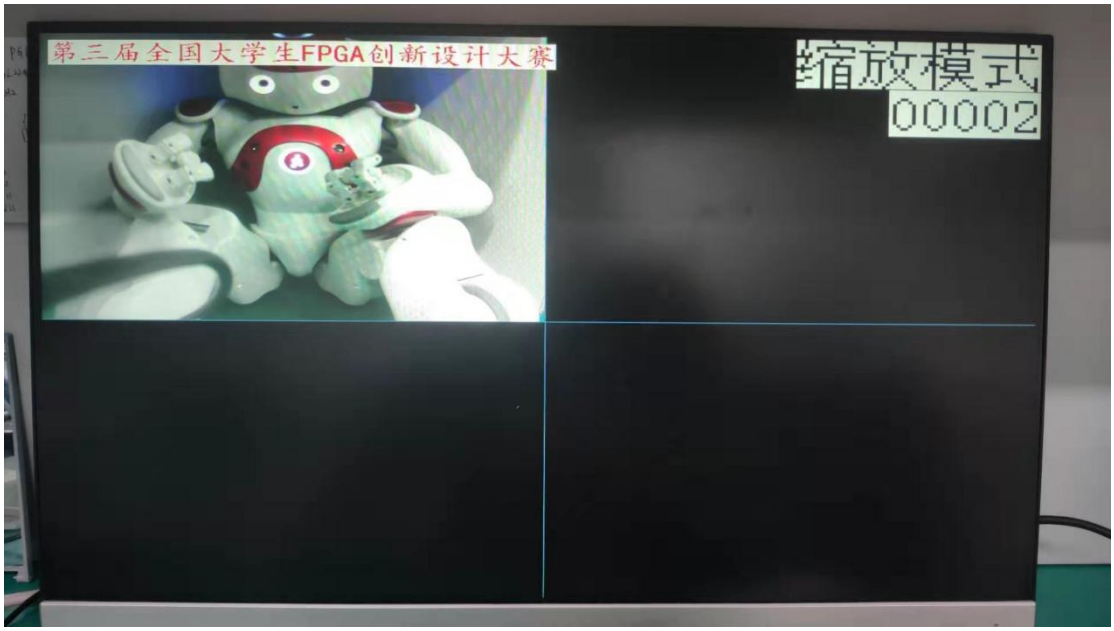


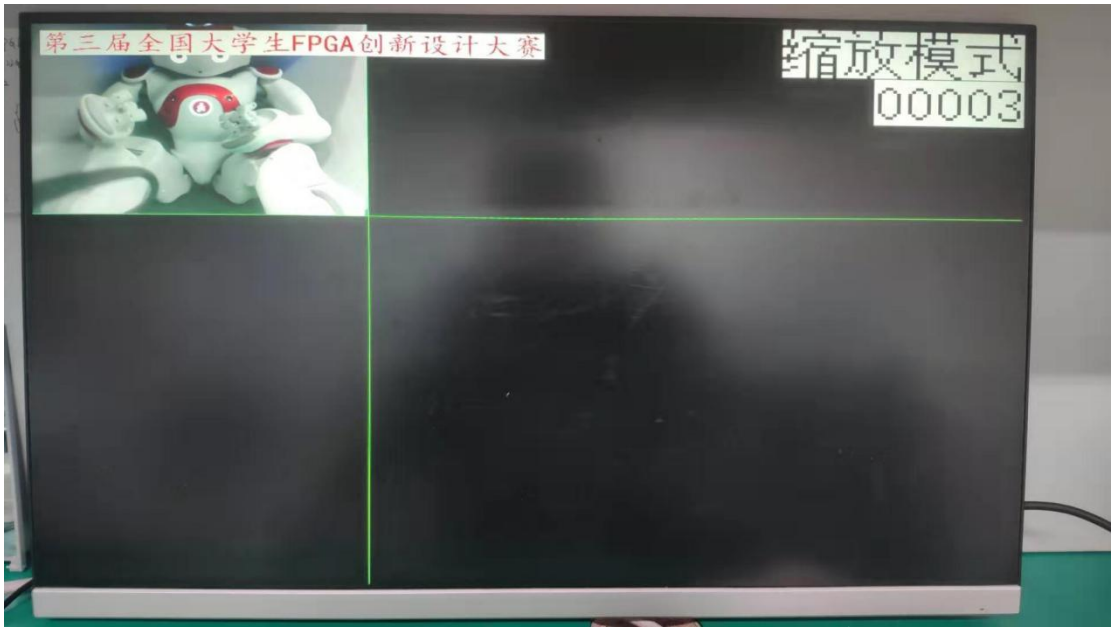
5.3.1,向右平移 125 个像素点



5.4 缩放模式

(在缩放模式下，显示的数字即为缩小的比例)





5.5 灰度模式

5.6 黑白模式

(在该模式下，显示的数字即为阈值的大小)


第三届全国大学生FPGA创新设计大赛
黑白显示
00065

第六部分 总结(Summary)

6.1 不足和改进

在我们团队的此次设计中，基本实现了视频图像的旋转，平移，缩放等处理。但还存在以下的不足之处。

1，在旋转图像的重建中，通过将 AXI 的突发长度设置为 1 来实现任意坐标像素数据进行读写，这种数据传输方式极大的浪费时间，增大了处理的延时，使得图像处理很慢，达不到一个很高的帧率。在以后的改进中，要采用随机读，突发写的方式来加快数据传输效率。

2，对于图像的算法和旋转，没有使用差值处理，结果图像的显示不是很光滑。在以后的改进中，要加上差值算法，使得图像显示效果更好。

6.2 致谢

首先感谢大赛组委会提供这次比赛的机会，其次感谢紫光同创公司对本次比赛提供的板卡支持。通过这次大赛，我们感受到了 FPGA 的魅力，以及 FPGA 在我们生活中的广泛应用，同时我们也在本次比赛中看到了当前 FPGA 厂商的优势，看到了在 ASIC 电路快速发展的今天，FPGA 自身的优势所在。通过本次比赛，我们更加了解了 FPGA 板卡的相关模块的特性，对 ROM,RAM,DDR3,以及摄像头，HDMI 的使用更加得心应手。对在计算机中常用的 CORDIC 算法进行了改进应用，在图像旋转的设计方面也更加的有经验。

总之通过这次比赛，我们感触良多，团队协作有了极大的提高，每个人的工程实践能力都有了很大的进步，希望以后可以更多的参加类似的比赛。

最后再次感谢大赛组委会以及紫光同创公司对本次比赛的大力支持。

——2019 年 11 月 22 日