1 基于 DDR3 的串口传图帧缓存系统设计实现

章节导读

本章将基于前面讲解的设计内容，继续讲解串口传图的基本内容。本章在 FPGA 学习内容中，处于立交桥的地位。本章的内容全面涵盖输入、缓存、输出，同时本章的内容又是基于 FPGA 片上图片缓存的内容深化。学完本章以后，既可以基于本章内容实现各种类型千变万化输入输出接口的替换设计，又可以在本章的数据接收与处理环节作文章，进行图像处理内容的理解与学习。

1.1 系统整体设计

本章我们将设计一个基于 DDR3 的串口传图帧缓存系统。该系统的整体设计框图如下。

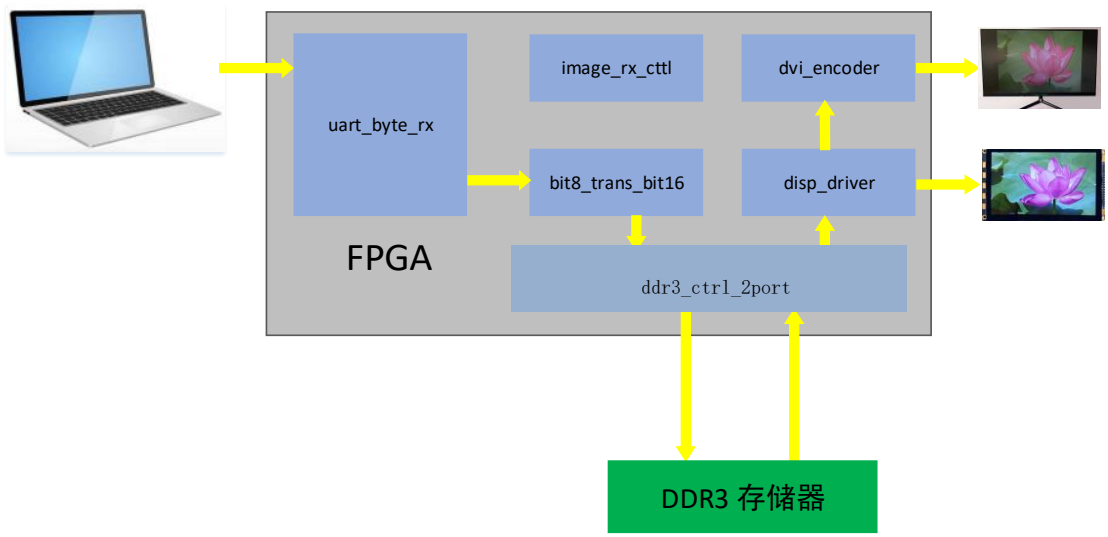

图 1-1 串口传图顶层系统设计框图

其中，

（1）uart_byte_rx 模块:负责串口图像数据的接收，该模块的设计前面章节已经有讲。

（2）bit8_trans_bit16 模块：将串口接收的每两个 8bit 数据转换成一个 16bit 数据（图像数据是 16bit 的 RGB565 的数据，电脑是通过串口将一个像素点数据分两次发送到 FPGA，FPGA 需将串口接收数据重组成 16bit 的图像数据），实现过程相对比较简单。

（3）disp_driver 模块：tft 屏显示驱动控制，对缓存在 DDR3 中的图像数据进行显示。

（4）ddr3_ctrl_2port 模块组：包含 wr_ddr3_fifo、rd_ddr3_fifo、fifo_to_axi4、axi4_to_fifo 以及 DDR_CONTROLLER 模块。完成采集的图像数据缓存。其中，wr_ddr3_fifo、rd_ddr3_fifo、fifo_to_axi4、axi4_to_fifo 已集成为 fifo_axi4_adapter。

（5）pll 模块：上述各个模块所需时钟的产生，使用 PLL IP。

（6）dvi_encoder 模块：用于将生成信号转换成 HDMI 输出信号作 HDMI 显示。

除去使用 IP 和前面章节讲过的模块（组）外，还需要设计 bit8_trans_bit16 模块和 image_rx_ctrl 模块。

1.2 接收控制模块设计(image_rx_ctrl)

为了对接收的信号进行分析和整理，我们设计了接收控制模块，主要内容是描述图像的行场同步信号生成。

```
module   image_rx_ctrl # (
    parameter DISP_WIDTH  = 800 ,
    parameter DISP_HEIGHT = 480
   )
   (
    input clk,
    input reset_p,
    input  image_data_valid,
    output reg [15:0]image_data_hcnt,
    output reg [15:0]image_data_vcnt,
    output reg image_data_hs,
    output reg image_data_vs,
    output reg frame_rx_done_flip
   );

     //generate image data hs or vs
  always@(posedge clk or posedge reset_p)
    if(reset_p)
      image_data_hcnt <= 'd0;
    else if(image_data_valid) begin
      if(image_data_hcnt == (DISP_WIDTH - 1'b1))
        image_data_hcnt <= 'd0;
      else
        image_data_hcnt <= image_data_hcnt + 1'b1;
    end
```

```
  always@(posedge clk or posedge reset_p)
    if(reset_p)
      image_data_vcnt <= 'd0;
    else if(image_data_valid) begin
      if(image_data_hcnt == (DISP_WIDTH - 1'b1)) begin
        if(image_data_vcnt == (DISP_HEIGHT - 1'b1))
          image_data_vcnt <= 'd0;
        else
          image_data_vcnt <= image_data_vcnt + 1'b1;
      end
    end
  //hs
  always@(posedge clk or posedge reset_p)
    if(reset_p)
      image_data_hs <= 1'b0;
    else if(image_data_valid && image_data_hcnt == (DISP_WIDTH - 1'b1))
      image_data_hs <= 1'b0;
    else
      image_data_hs <= 1'b1;
  //vs
  always@(posedge clk or posedge reset_p)
    if(reset_p)
      image_data_vs <= 1'b0;
    else if(image_data_valid && image_data_hcnt == (DISP_WIDTH - 1'b1) &&
            image_data_vcnt == (DISP_HEIGHT - 1'b1))
      image_data_vs <= 1'b0;
    else
      image_data_vs <= 1'b1;

  always@(posedge clk or posedge reset_p)
    if(reset_p)
      frame_rx_done_flip <= 1'b0;
    else if(image_data_valid && image_data_hcnt == (DISP_WIDTH - 1'b1) &&
            image_data_vcnt == (DISP_HEIGHT - 1'b1))
      frame_rx_done_flip <= ~frame_rx_done_flip;
endmodule
```

在该模块之中，完成了图像行同步信号描述及计数，图像场同步信号描述及计数，图像接收完成信号描述。通过这些信息，就可以定位图像传输的进度。同时，如果图像传输完成，则给出传输完成的 led 点亮信号。

1.3 位宽转换模块设计

bit8_trans_bit16 该模块主要功能是将串口传输过来（一次传 8bit）的每两个

8bit 数据拼接成一个 16bit 图像数据（RGB565）。功能相对比较简单。该模块的接口图及接口功能描述如下。

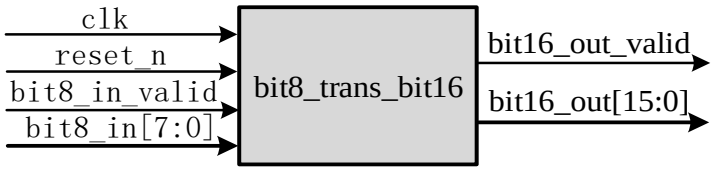

图 1-2 bit8_trans_bit16 模块接口图

表 1-1 bit8_trans_bit16 模块接口功能描述

接口名称	I/O	功能描述
clk	I	模块工作时钟
reset_n	I	模块复位，低有效
bit8_in[7:0]	I	8bit 数据输入
bit8_in_valid	I	8bit 数据有效标识
bit16_out[15:0]	O	转换后 16bit 数据输输出
bit16_out_valid	O	转换后 16bit 数据有效标识

模块主要信号设计的时序要求如下。对输入的每两个 8bit 数据进行拼接，并产生一个拼接后数据输出有效标识信号。拼接后的数据是前一个数据在高字节位置，后一个数据在低字节位置。这个主要是串口传图上位机软件发送是是先发送 16bit 图像数据的高字节，后发低字节，FPGA 上这样处理为了保证拼接后图像数据的正确性。

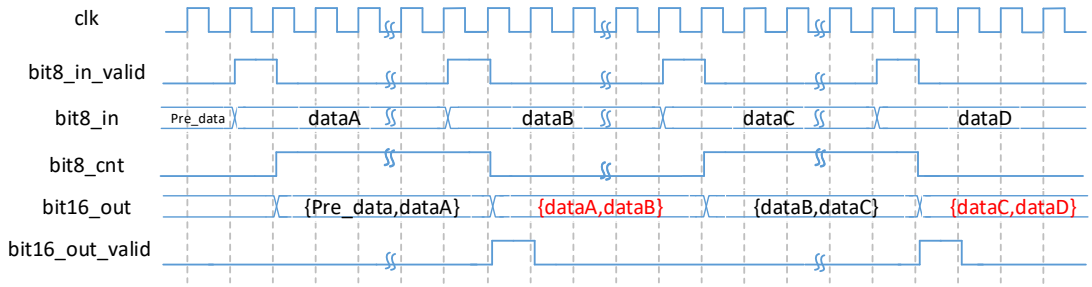

图 1-3 bit8_trans_bit16 数据转换时序图

有了时序图，代码设计就比较简单，具体代码如下。

```
module bit8_trans_bit16(
  input             clk,
  input             reset_p,

  input      [7:0] bit8_in,
  input            bit8_in_valid,

  output reg [15:0]bit16_out,
  output reg       bit16_out_valid
);
```

```
  reg bit8_cnt;

  always@(posedge clk or posedge reset_p)
  begin
    if(reset_p)
      bit8_cnt <= 1'b0;
    else if(bit8_in_valid)
      bit8_cnt <= bit8_cnt + 1'b1;
    else
      bit8_cnt <= bit8_cnt;
  end

  always@(posedge clk or posedge reset_p)
  begin
    if(reset_p)
      bit16_out <= 16'h0000;
    else if(bit8_in_valid)
      bit16_out <= {bit16_out[7:0],bit8_in};
    else
      bit16_out <= bit16_out;
  end

  always@(posedge clk or posedge reset_p)
  begin
    if(reset_p)
      bit16_out_valid <= 1'b0;
    else if(bit8_in_valid && bit8_cnt)
      bit16_out_valid <= 1'b1;
    else
      bit16_out_valid <= 1'b0;
  end
endmodule
```

通过以上设计，位宽转换模块就设计完成。

1.4 系统仿真验证与板级测试

各个子模块设计完成后，顶层的设计就相对容易些，根据整体设计框图对子模块端口信号进行连接即可，具体代码参见本章工程源码。顶层的仿真与 fifo_axi4_adapter 模块仿真类似，这里就不做详细讲解，读者可以参考本章工程源码完成顶层的仿真。注意本次设计的顶层仿真耗时可能较长，实测在 1ms 之后才逐渐开始出现 TFT_rgb 数据。

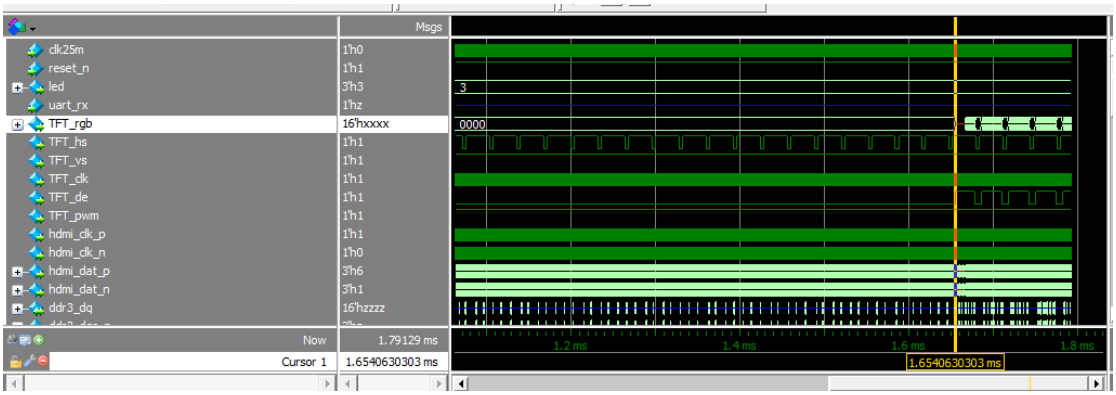

图 1-4 顶层仿真（局部）

1.4.1 管脚绑定

根据 AC201-SA5Z-50D0 开发板提供的管脚绑定表，我们对 TFT 显示屏和串口的管脚，直接进行绑定即可。

由于涉及到 DDR3 的工程，管脚数量都不少，受制于篇幅，读者可以自己参考教程配套的工程代码完成管脚绑定。本工程涉及到绑定的部分，主要为时钟、复位基础管脚，有 DDR 相关管脚，串口相关管脚，TFT 相关管脚等几个部分。

1.4.2 串口传图工程的 TFT 显示

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，准备进行上板验证。由于本工程涉及管脚数量较多，我们也就不以列表的形式展示本章例程的管脚绑定了。如需参考，读者可以直接从例程中赋值本工程的管脚绑定 upc 文件到自己设计的工程中对自己的设计进行验证。

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，对应硬件连接如下图所示:

图 1-5 串口传图开发板连接图

连接好开发板后，下载生成的 Bit 文件。

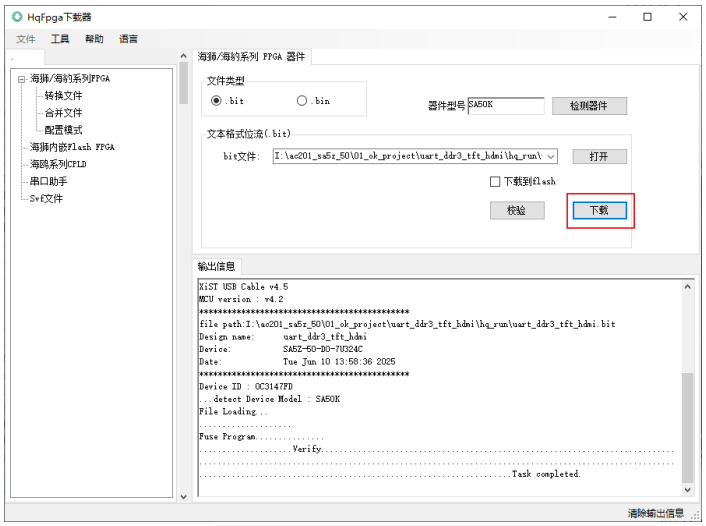

图 1-6 下载串口传图 bit 文件

下载完成后，开发板上 LED0~LED1 会亮，TFT5.0 寸屏上显示花屏状态。如下图所示。

图 1-7 下载完程序后产生的碎花屏效果

出现这种花屏是因为这个时候，DDR3 中并未写入数据，显示的数据是不可知的一些数据。LED0 和 LED1 分别表示的是 DDR 控制器内时钟锁相环的 locked 信号和 DDR 初始化校准完成信号的状态，亮表示这个两个信号均变为高电平，说明 DDR 已经正常完成初始化和校准操作。接下来通过小梅哥串口传图工具向 FPGA 传输图片数据。小梅哥串口传图工具可在论坛下载。

图 1-8 启动串口传图工具

双击打开串口传图工具，通过点击“打开图片”按钮设置图片存放路径；图片宽度和高度设置成与显示屏分辨率一致（TFT5.0 寸屏是 800*480）。下图是待传的图片。

图 1-9 需上传的图片

关于图片的信息可通过右键图片查看其属性，在属性的详细信息窗口可以看到图片大小，位深度等信息。注：这里提供的上位机要求图片为位深度为 24

或 16 的 bmp 格式图片，图片的宽度和高度需要为 800*480（插 5 寸屏情况下）或 480*272（插 4.3 寸屏情况下）。

图 1-10 需传图片的属性信息

串口波特率设置与 FPGA 串口接收波特率一致（FPGA 串口接收波特率是 1562500bps），串口端口号根据实际连接电脑的串口号进行设置（这里使用的是 COM14），设置好传图工具后，点击“连接设备”。

图 1-11 软件连接图片

连接成功后，“连接设备”会变成“断开设备”，通过点击“发送图片”按钮开始发送图片数据。

图 1-12 设置完图片信息后向对应的 com 端口号发送图片

传图过程中，可以看到 TFT 屏上开始显示发送的图片。图片传送完成后，TFT 屏显示效果如下。

图 1-13 串口传图完成后的图片效果

以上就完成了串口传图 TFT 显示的工程设计。

1.4.3 串口传图工程添加 HDMI 显示

该显示内容还可以通过 HDMI 接口输出到 HDMI 显示器上。当然，添加 HDMI 显示后，TFT 显示也可以保留。

为了实现以上目标，我们可以对工程作如下改进：

1. 在锁相环中，添加 165M 时钟信号输出。

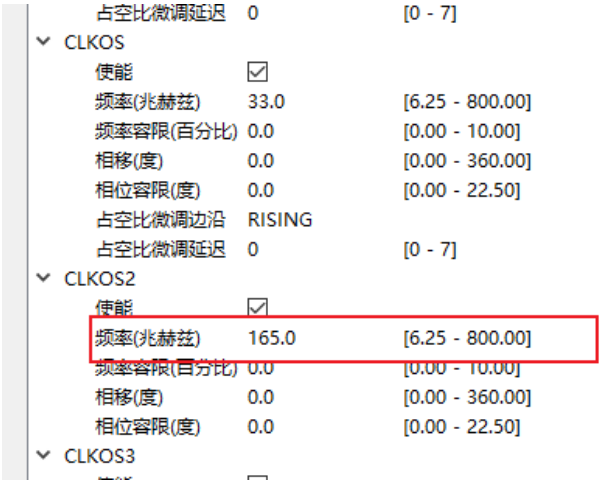

图 1-14 锁相环添加 165MHz 输出频率

2. 在端口处添加 HDMI 输出管脚，并在 XDC 文件或管脚配置界面进行

HDMI 管脚绑定。

```
//hdmi1 interface
output          hdmi1_clk_p   ,
output          hdmi1_clk_n   ,
output [2:0]    hdmi1_dat_p   ,
output [2:0]    hdmi1_dat_n   ,
output          hdmi1_oe
```

3. 添加 HDMI 接口例化：

```
//HDMI
dvi_encoder dvi_encoder(
    .pixelclk    (pixelclk    ),
    .pixelclk5x  (pixelclk5x  ),
    .rst_p       (g_rst_p     ),
    .blue_din    (disp_blue   ),
    .green_din   (disp_green  ),
    .red_din     (disp_red    ),
    .hsync       (disp_hs     ),
    .vsync       (disp_vs     ),
    .de          (disp_de     ),
    .tmds_clk_p  (hdmi_clk_p ),
    .tmds_clk_n  (hdmi_clk_n ),
    .tmds_data_p (hdmi_dat_p ),
    .tmds_data_n (hdmi_dat_n )
);
```

4. RGB565 和 RGB888 的转换

```
assign hdmi1_oe = 1'b1;
wire [7:0] disp_red;
wire [7:0] disp_green;
wire [7:0] disp_blue;


assign disp_red = {TFT_rgb[15:11],3'b0};
assign disp_green = {TFT_rgb[10:5],2'b0};
assign disp_blue = {TFT_rgb[4:0],3'b0};

wire[15:0] image_data;
assign wrfifo_din  = image_data;
assign wrfifo_wren = image_data_valid;

wire disp_hs = TFT_hs;
wire disp_vs = TFT_vs;
wire disp_de = TFT_de;
```

5. 添加 HDMI 管脚绑定

Enabled	TFT_pwm	← OUTPUT	P4 bank12	...	LVCMOS33	NONE
Enabled	hdmi_clk_p	← OUTPUT	T3 bank12	...	LVCMOS33	NONE
Enabled	hdmi_clk_n	← OUTPUT	R3 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_p[2]	← OUTPUT	V4 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_p[1]	← OUTPUT	T6 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_p[0]	← OUTPUT	U3 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_n[2]	← OUTPUT	R7 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_n[1]	← OUTPUT	U4 bank12	...	LVCMOS33	NONE
Enabled	hdmi_dat_n[0]	← OUTPUT	N6 bank12	...	LVCMOS33	NONE

图 1-15 HDMI 相关管脚绑定

6. 完成以上配置后，在前面给出的线缆连接图中，添加开发板 HDMI 接口和液晶显示器的连接。

7. 将改造好的工程下载到 FPGA 开发板，HDMI 显示如下：

图 1-16 串口传图 TFT 和 HDMI 联合显示实验效果

至此，串口传图帧缓存系统在 ACX750 开发板上的设计及验证就成功完成了。

1.5 总结

本章讲解了基于 DDR3 的串口传图帧缓存系统的设计与实现方法。通过本章的学习，各位读者可以更深入的对输入控制器，缓存控制器，输出控制器的设计有一个更为全面宏观的认识，在这些控制器协调工作时，既需要关注数据在流动中形态的不同，又需要关注数据收发控制信号和不同时钟频率信号的协调。