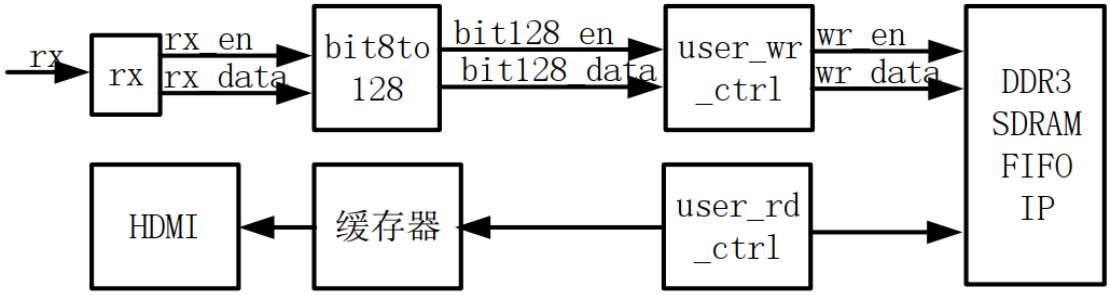
高级课程7 UART+DDR3+HDMI 联合实现图片显示

一、练习内容

从UART读取数据，写入DDR3内存中，再从DDR3内存中读取数据，在HDMI显示器上显示出来。

二、系统框图



上图的系统框图中，从DDR3读取数据，然后显示在HDMI显示器的这部分已经搞定，这次练习主要从考虑将uart读取的数据写入DDR3的过程，其中主要是实现8bit到128bit模块的实现。

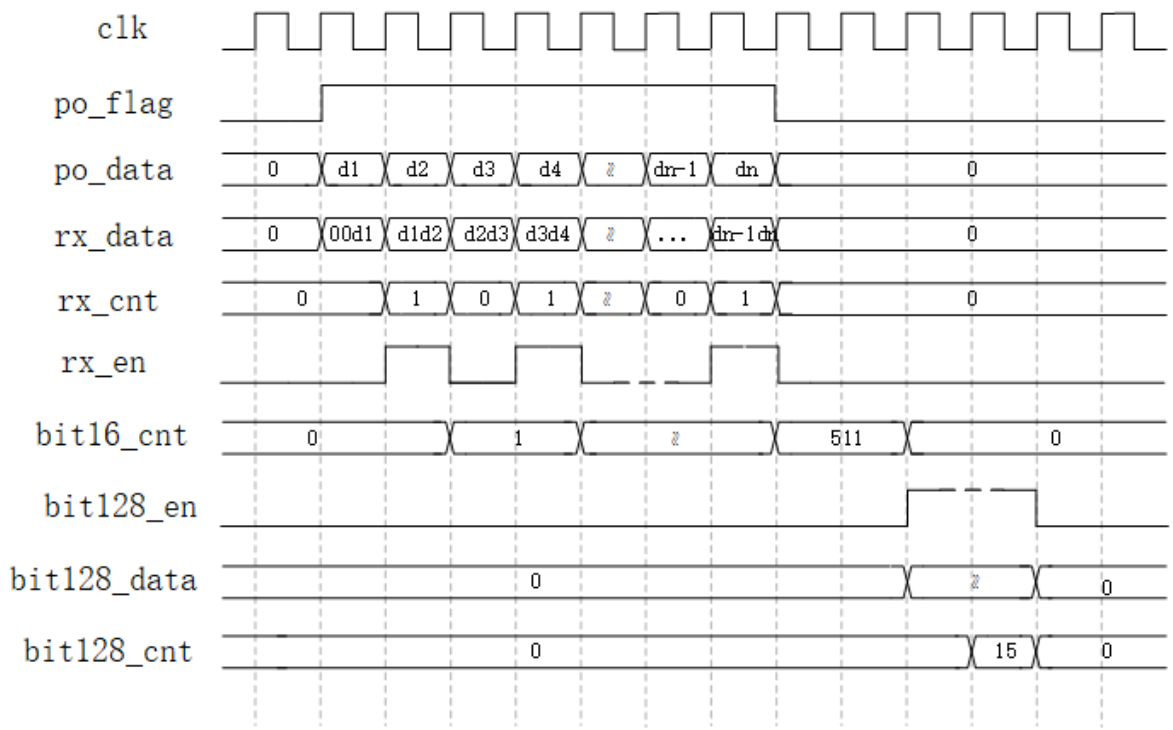
三、设计分析

1.时序分析

上面已经说了，这部分主要有两个模块需要实现。

首先，对于串口接收，以前已经实现了，这里直接拿过来用就行了。

对于bit8转bit128模块的实现，这里有一点需要注意，就是实际fifo写入的数据为16bit，读取的数据位数则为128bit，这样的话，需要先将数据转换为16bit，再写入fifo中，具体的时序如下图所示。



2.实现分析

这里需要调用一个fifo模块，将16bit的数据写入，当fifo中的数据达到128*64/16(512)时，启动fifo数据的读取，这里读取的128bit的数据，即128bit的数据达到64突发长度后，就拉低写使能。

为此，这里有一点需要注意。首先，fifo 16bit数据写入的深度必须大于512，这里设置为其两倍，即1024，这样的话，fifo 128bit数据读取的深度则是128.

四、练习步骤

按照上面的分析，实现相应的代码即可。

1.代码编写

按照时序波形编写代码即可，为了保证时序的准确性，这里使用modelsim进行仿真，保证其准确性。

简单起见，这里只考虑数据写入DDR3的过程，其余部分不需要考虑。

这里对于仿真的代码实现贴出来查看。

```
    task gen_wr_data;
        integer i;
        begin
            @ (negedge rst);
            repeat (100) @ (posedge clk);
            wr_en = 1;
            for (i = 0; i < 512; i = i + 1)
            begin
                wr_data = i[7:0];
                @ (posedge clk);
            end
            wr_en = 0;
            @ (posedge clk);
            wr_en = 1;
            for (i = 0; i < 512; i = i + 1)
```

```
            begin
                wr_data = i[7:0];
                @ (posedge clk);
            end
            wr_en = 0;
        end
    endtask
```

2.ila时序抓取

这里通过添加ila来抓取信号，主要是要查看bit8to128模块的时序是否正确，因此需要将其中的wr_en和po_flag等信号拉出来查看。

其次，若前面部分抓取信号正确，但却没有正常的显示的话，就要考虑user_wr_ctrl模块是否正确，需要将其中的p2_wr_en、p2_wr_empty和p2_cmd_en、p2_cmd_empty等信号拉出来进行查看。

抓取这些信号基本上就够了。

3.查看结果

(1) matlab生成像素文件

需要先用matlab将1024*768的图像文件中的像素提取出来，然后再进行显示。提取实现代码如下。

```
clear all;
RGB = imread('p1.jpg');
[ROW,COL,N] = size(RGB);
fid = fopen('grayimage1.txt','w+');
for i = 1:ROW
    for j = 1:COL
    RG=bitand(RGB(i,j,1),248) + bitshift(RGB(i,j,2),-5);%R[7:3]G[7:5]
    GB=bitshift( bitand(RGB(i,j,2),28),3) +
bitshift(RGB(i,j,3),-3);%G[4:2]B[7:3]
    fprintf(fid,'%02x %02x ',RG,GB);
    end
end
fclose(fid);
```

(2) 查看显示结果

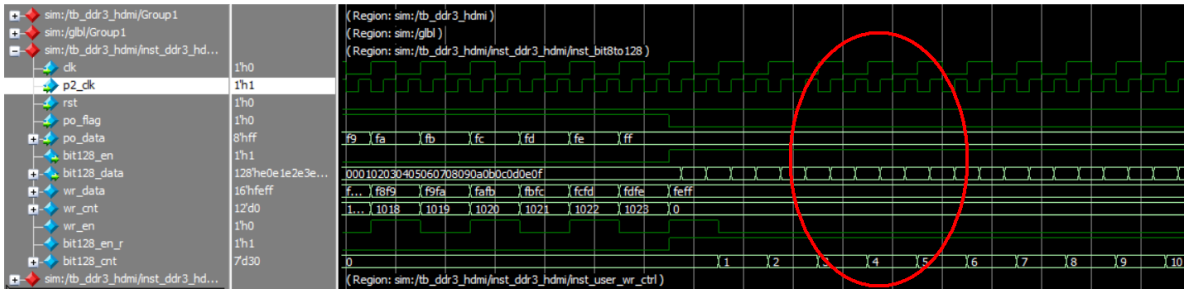
在显示屏上显示出来结果。

五、实际波形仿真

1.Modelsim仿真

(1)查看时序是否正确

这个主要查看bit8to128模块的时序，仔细观察波形。



查看波形，发现读数据的时钟出现了错误，导致整个过程出错，计数偏少，而实际上fifo中的数据已全部读出。

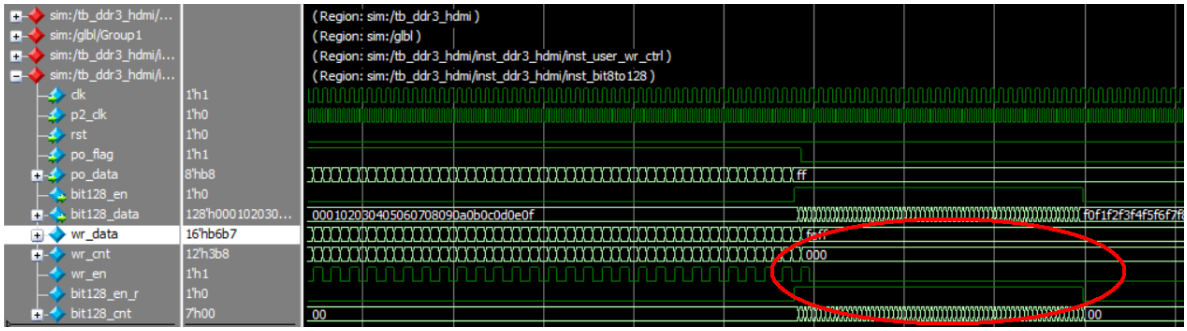
需要进行时钟的匹配，修改时钟和相应的标志位。

```
75  //bit128_en
76  always @(posedge p1_clk)
77  begin
78      if (rst == 1'b1)
79          bit128_en_r <= 1'b0;
80      else if (wr_cnt >= BIT8_END_CNT)
81          bit128_en_r <= 1'b1;
82      else if (bit128_en == 1'b1 && bit128_cnt == BIT128_END_CNT)
83          bit128_en_r <= 1'b0;
84  end
```

因为这里只需要个数足够了，不需要说一定要在写状态，保证一定能够读到数据。然后，再次进行仿真。

(2)查看数据是否写入

这个则主要观察user_wr_ctrl的时序，主要观察数据。

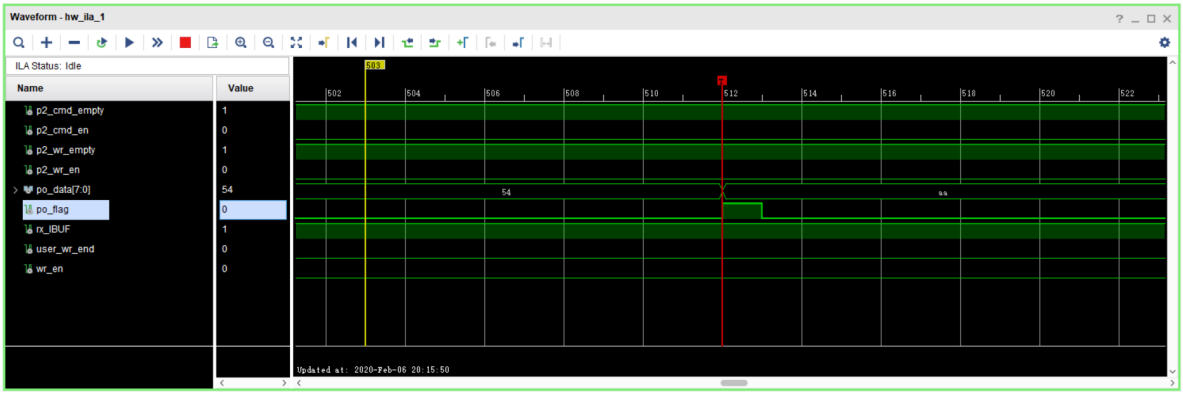


波形正确，仿真这部分搞定。

2.ILA抓取信号

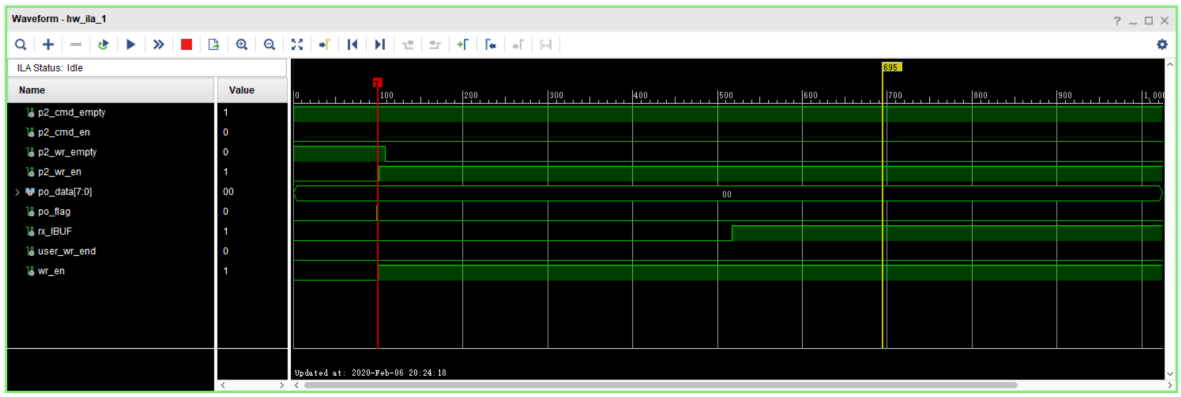
（1）bit8to128模块外部信号抓取

首先，查看是否uart是否读取的正确的数据。需要抓取的信号为po_flag和po_data，以发送0xAA为例进行查看。



正确检测到po_flag的上升沿，同时检测到相应的数据po_data为0xAA，所以uart读取信号正确。

然后，bit8to128模块的wr_en是否正常输出。需要抓取的信号是wr_en，发送整幅图像转化后的数据。

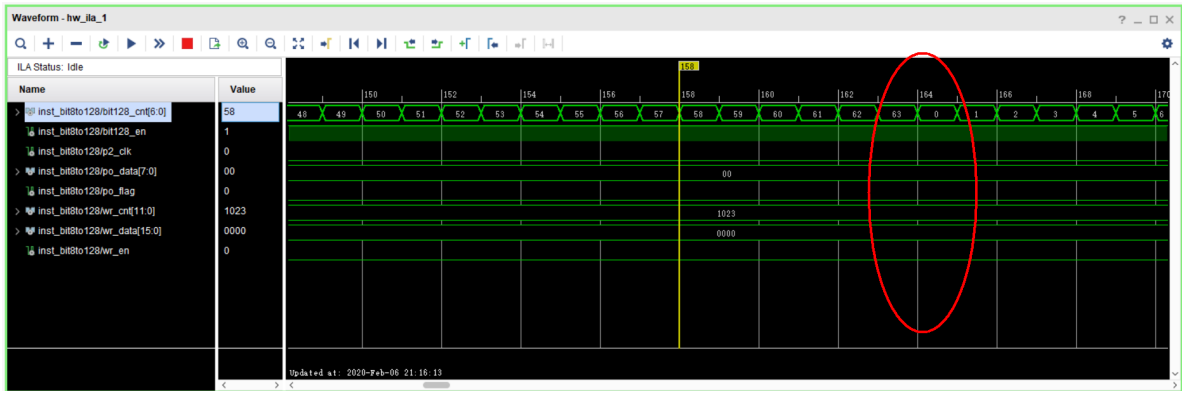


发现虽然读取了数据，但是读取的数据长度不大正确，代码部分可能还存在问题。

wr_en的时长明显不对，应该到63就要清零了，才对啊。这部分时序存在问题，需要仔细考量。

（2）bit8to128模块内部信号抓取

既然，暂时没有发现问题，那就从内容查看具体的问题，看是哪方面出现了问题。这里直接抓取bit8to128模块的内部信号。



发现问题，bit128_cnt一直在往上加，然后清零，这样不停的操作。这明显就有问题，使得读取相同的数据过多。再返回来查看代码，发现是bit128拉高的节点不对，这里需要进行修改。然后，重新编译查看结果。

```
75  //bit128_en
76  always @(posedge p2_clk)
77  begin
78      if (rst == 1'b1)
79          bit128_en_r <= 1'b0;
80      else if (wr_cnt >= BIT8_END_CNT)
81          bit128_en_r <= 1'b1;
82      else if (bit128_en == 1'b1 && bit128_cnt == BIT128_END_CNT)
83          bit128_en_r <= 1'b0;
84  end
```

上述方式又引进了新问题，即bit128_en一直没有拉高，这里使用存在一个问题，就是读写时序不一致的问题，导致数据的判定也一定程度上存在问题。

故重新修改为如下的方式实现。

```
77  //bit128_en
78  always @(posedge p2_clk)
79  begin
80      if (rst == 1'b1)
81          bit128_en_r <= 1'b0;
82      else if (rd_data_count > BIT128_END_CNT)
83          bit128_en_r <= 1'b1;
84      else if (bit128_en == 1'b1 && bit128_cnt == BIT128_END_CNT)
85          bit128_en_r <= 1'b0;
86  end
```

其中rd_data_count是fifo读取的数据，而不是自己累加的数据，会实时更新，这样基本上就搞定了。

六、总结与讨论

1.关于IP核的调用原则

- 能使用IP核尽量使用IP核

  写好的fifo为啥不用，必须要用

- 能调用内部信号解决就调用内部信号解决

  对于fifo来说，充分利用fifo的wr_count和rd_count解决问题

- IP核的资源是有限的，需要精确计算占用资源，防止浪费

  对于fifo来说，fifo的数据深度需要尽可能地节省

2. 调试技巧

- 各个击破，逐步解决
- 整体上把控，细微处入手
- 尽可能使用软件仿真，因为调用ILA花费时间过长