

高级练习6 基于 A7 的 DDR3 与 HDMI 连接器

笔记本：FPGA练习

创建时间：2020/2/5 8:55

更新时间：2020/2/6 10:59

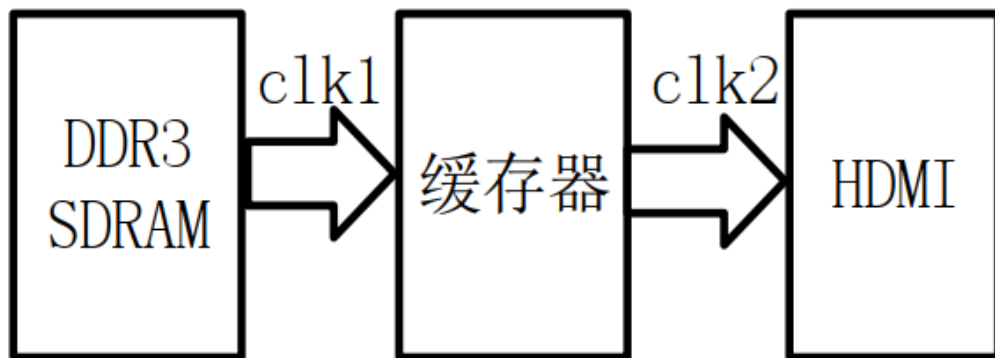
作者：2322900041@qq.com

高级练习6 基于 A7 的 DDR3 与 HDMI 连接器

一、练习内容

将DDR3的内容读出，经过HDMI显示出来，图像的分辨率是1024*768@60Hz

二、系统框图



三、设计分析

1. 缓存器的必要性

DDR3读取数据的时序和HDMI显示时序的不一致性，使得两者的数据读取存在问题，这样必须添加中间缓存器起到连接的作用，保证数据的一致性。

2. fifo缓存器三大问题

（1）问题分析

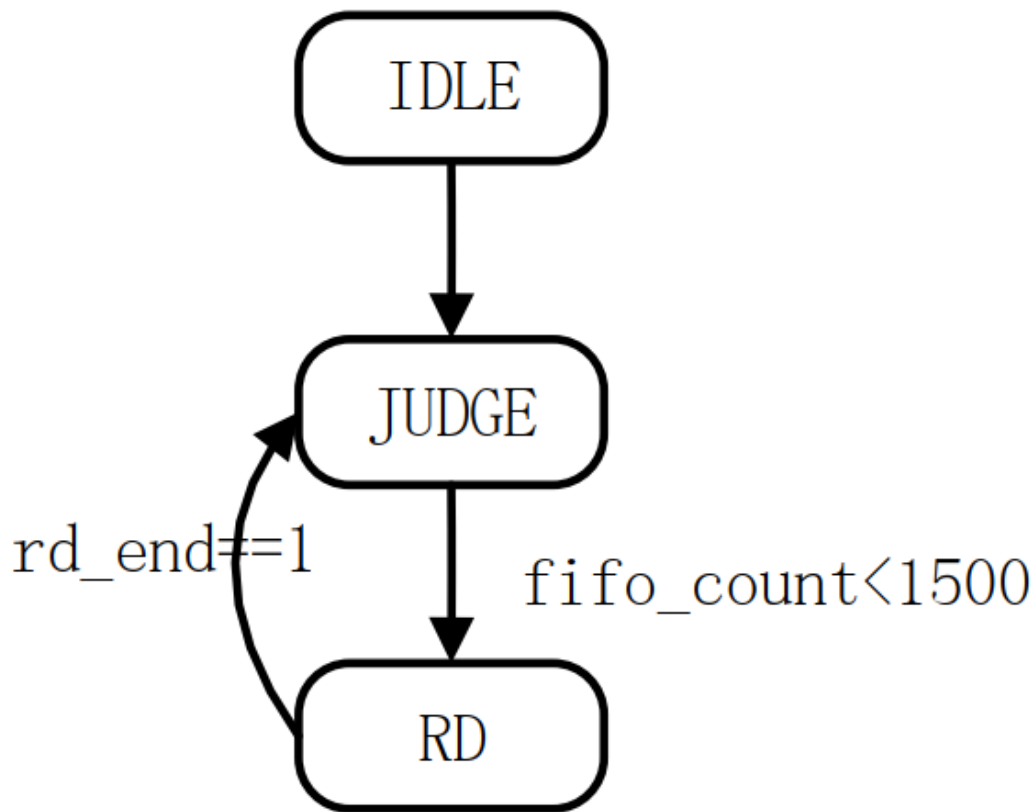
- 等待DDR3内存中的数据达到1.5行的显示要求再读取数据
- 必须在显示区域读取数据
- 从屏幕的显示的初始位置开始读取数据

（2）解决办法

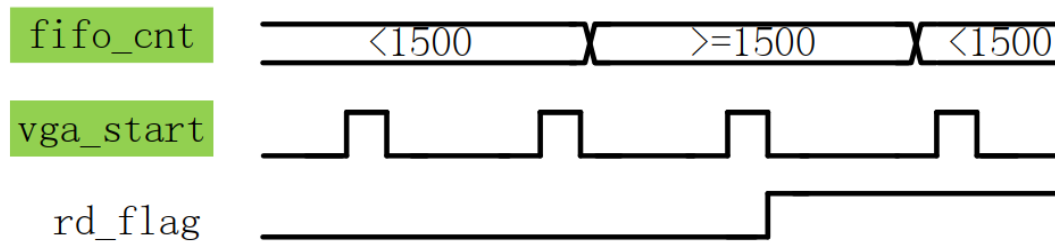
- 时刻监测fifo读取数据量的大小，当前满足读取1.5行的要求时，开始读取数据
- 等待一帧图像的开始和图像有效时开始读取数据

3. 时序分析

（1）fifo写入



(2) fifo读取



四、练习步骤

1. HDMI显示

(1) VGA时序修正

根据时序要求，修改关键的时序点

```
//1024*768@60Hz
parameter      H_SYNC_TIME      =136;
parameter      H_BACK_PORCH     =160;
parameter      H_LEFT_BORDER    =0;
parameter      H_ACT_START      =H_SYNC_TIME + H_BACK_PORCH + H_LEFT_BORDER;
parameter      H_ACTIVE_TIME    =1024;
parameter      H_ACT_END        =H_ACT_START + H_ACTIVE_TIME;
parameter      H_TOTAL_TIME     =1344;
parameter      V_TOTAL_TIME     =806;
parameter      V_SYNC_TIME      =6;
parameter      V_BACK_PORCH     =29;
parameter      V_TOP_BORDER     =0;
parameter      V_ACT_START      =V_SYNC_TIME + V_BACK_PORCH + V_TOP_BORDER;
parameter      V_ACTIVE_TIME    =768;
parameter      V_ACT_END        =V_ACT_START + V_ACTIVE_TIME;
```

(2) 关键时钟和使能信号时序控制

这里需要增加的只是vga_start信号，这里使用计数的最开始部分作为每幅图像的开始。

```
185 assign vga_start = h_sync_start_flag & v_sync_start_flag;
```

时钟信号，这里需要注意有3个时钟，分别是

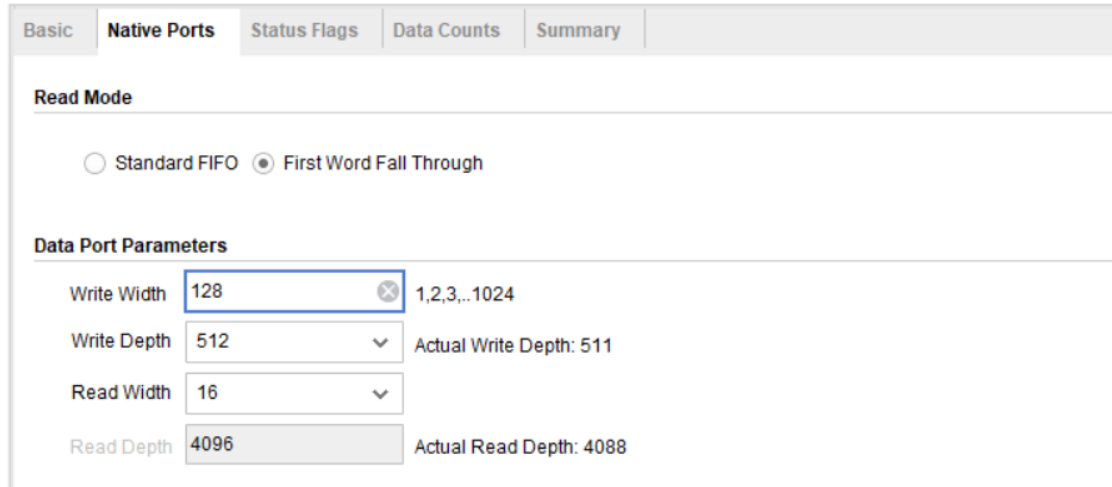
- hdmi输入时钟clk1x: 65MHz
- hdmi输入5倍时钟clk5x: 325MHz
- 读取数据时钟p1_clk

Timing Name	= 1024 x 768 @ 60Hz;			
Hor Pixels	= 1024;	// Pixels		
Ver Pixels	= 768;	// Lines		
Hor Frequency	= 48.363;	// kHz	= 20.7 usec	/ line
Ver Frequency	= 60.004;	// Hz	= 16.7 msec	/ frame
Pixel Clock	= 65.000;	// MHz	= 15.4 nsec	± 0.5%
Character Width	= 8;	// Pixels	= 123.1 nsec	

2. DDR3数据读取

(1) 调用fifo设置

设置输入数据的宽度为128，采样深度为512，保证满足一行以上的数据进行读取。



Basic Native Ports Status Flags Data Counts Summary

Read Mode

☐ Standard FIFO ☒ First Word Fall Through

Data Port Parameters

Write Width: 128 (1,2,3,...,1024)

Write Depth: 512 (Actual Write Depth: 511)

Read Width: 16

Read Depth: 4096 (Actual Read Depth: 4088)

(2) rd_flag问题修正

对于rd_flag来说，只要拉低一次就够了，后面会按照正常的顺序进行下去。

```
77 //rd_flag
78 always @(posedge vga_clk)
79 begin
80     if (rst == 1'b1)
81         rd_flag <= 1'b0;
82     else if (state == JUDGE && rd_data_count >= RD_END_CNT && vga_start == 1'b1)
83         rd_flag <= 1'b1;
84     else if (state == RD)
85         rd_flag <= 1'b0;
86 end
```

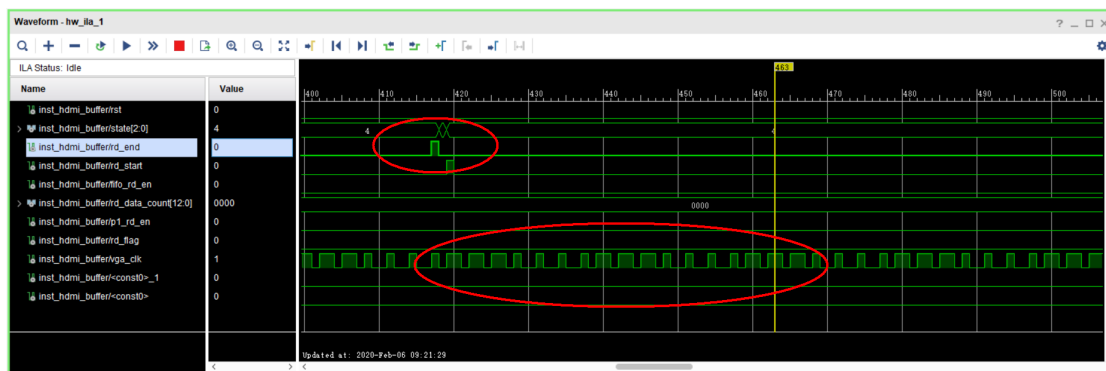
五、实际波形仿真

1. 查看ila波形

(1) 查看hdmi_buffer中的部分信号

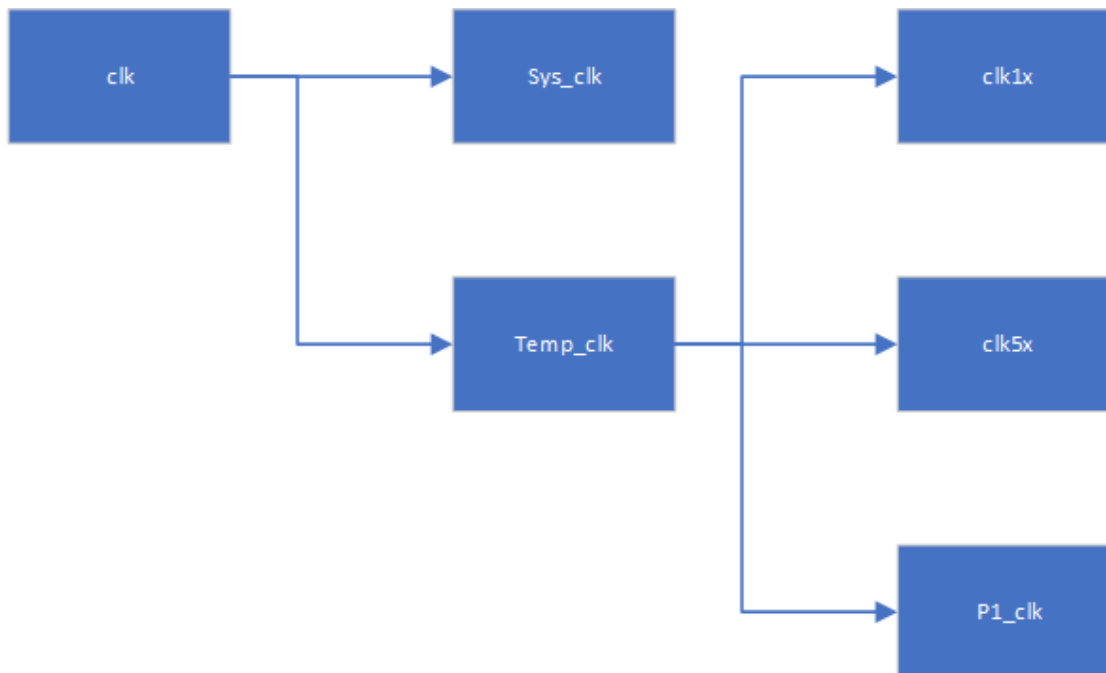
波形如下图所示，主要存在俩方面的问题

- DDR3时钟信号不一致
- rd_end信号不对



下面分别对问题进行解决。

首先是DDR3的时钟不一致问题，采用下面的框图进行解决，使用了两个PLL。



需要注意的是第二个PLL的时钟源设置为buffer_g，如下图所示。

Input Frequency(MHz)		Jitter Options	Input Jitter	Source
100.000	10.000 - 800.000	UI	0.010	Global buffer
100.000	60.000 - 144.000		0.010	Single ended clock capabl...

第二个问题是rd_end信号不对，发现连接出现了问题。

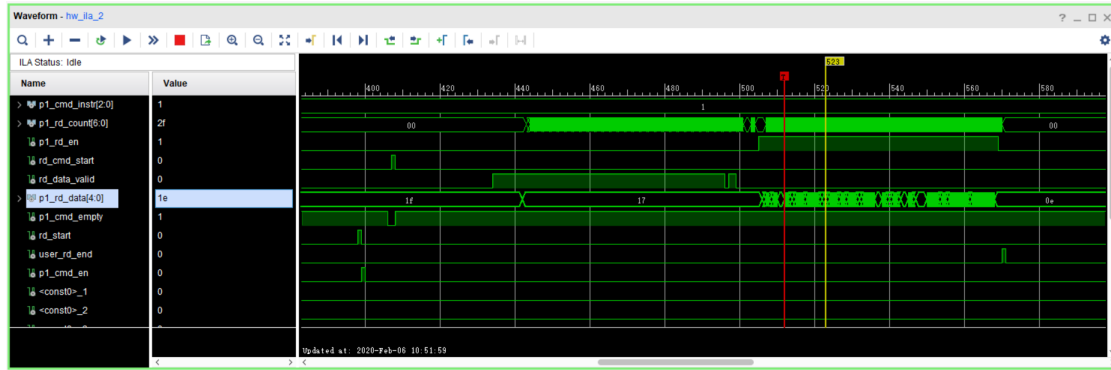
应该是连接方面的问题

```

362  inst_hdmi_buffer (
363      .p1_clk      (p1_clk),
364      .vga_clk     (clk1x),
365      .rst         (ui_clk_sync_rst|(~init_calib_complete)),
366      .p1_rd_en    (p1_rd_en),
367      .p1_rd_data  (p1_rd_data),
368      .rd_end      (rd_end),
369      .vga_start   (vga_start),
370      .vga_de      (vga_de),
371      .rd_start    (rd_start),
372      .fifo_rd_data ({red, green, blue})
373  );
  
```

(2) 测试结果

基本上没有什么问题，搞定

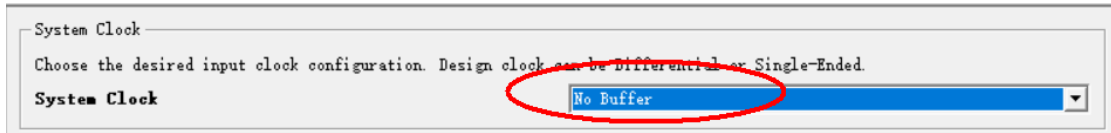


六、总结与讨论

1. 部分程序修正

要达到正常使用的要求，需要对本程序进行简单的修正

(1) DDR3配置修正



sys_clk时钟修改为no buffer输入，即内部产生，这样就不需要再添加额外的时钟信号了。

(2) 读写地址清零修正

需要按照显示图形的要求，使得地址写完一帧后，重新回到开始地址。

```
//1024*768@60Hz
parameter          START_ADDR = 0,
                    STOP_ADDR  = 785920,    //1024*768-512 =
512
                    ADDR_ADDR  = 512;

//128*64/16 = 512

//p2_cmd_addr
always @(posedge clk)
begin
```

```

        if (rst == 1'b1)
            p2_cmd_addr_r <= START_ADDR;
        else if (p2_wr_en == 1'b0 && p2_wr_en_r == 1'b1 && p2_cmd_addr_r ==
STOP_ADDR)
            p2_cmd_addr_r <= START_ADDR;
        else if (p2_wr_en == 1'b0 && p2_wr_en_r == 1'b1)                //
下降沿检测
            p2_cmd_addr_r <= p2_cmd_addr_r + {p2_cmd_b1, 3'd0};
    end

```

(3) 亚稳态消除（时序违例）

首先，打俩拍，解决两者时序的不一致问题。

```

always @(posedge clk)
begin
    p2_wr_empty_r1 <= p2_wr_empty;
    p2_wr_empty_r2 <= p2_wr_empty_r1;
    p2_wr_empty_r3 <= p2_wr_empty_r2;
end

```

然后，考虑用大俩拍后的信号来判定结束信号。

```

//user_wr_end
always @(posedge clk)
begin
    if (rst == 1'b1)
        user_wr_end_r <= 1'b0;
    else if (p2_wr_empty_r2 == 1'b1 && p2_wr_empty_r3 == 1'b0)
//上升沿检测
        user_wr_end_r <= 1'b1;
    else
        user_wr_end_r <= 1'b0;
end

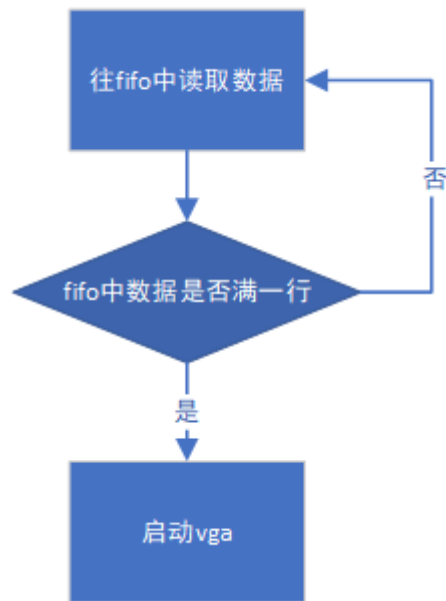
```

至此，准备工作完成。

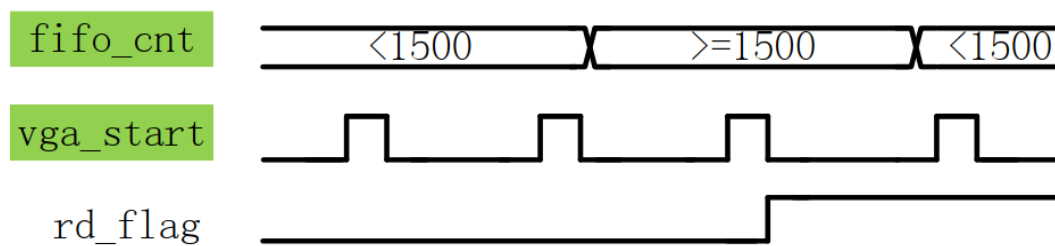
2. vga时序控制的两种方式

vga时序控制特别需要注意显示的开始部分必须是一幅图像的开始，即fifo中的数据往外读取时，要求vga时序控制从初始位置起步。这样的话有两种实现方式，一是外部控制vga，即fifo中存满一行以上的数据时，才开始使能vga模块启动，再次之前，vga一直处于复位状态，这个是很容易理解的；二是内部控制vga，每次到达初始位置，vga自己发送vga_start标志表示显示的开始，在外部控制中则需要在fifo中存满一行，同时vga_start拉高时，再开始读取数据。

(1) 外部控制vga



(2) 内部控制vga



需要注意的是，两种方式是第一次正确固定位置就行了，后面自然会按照正常的顺序读取数据，无需再次进行考虑。