

笔记本: FPGA练习
创建时间: 2020/2/3 19:40
作者: 2322900041@qq.com

高级练习5 用户端实现 DDR3 SDRAM 写、读控制

在已经实现的DDR3控制器的基础上，实现用户的读写控制。

The diagram illustrates the internal architecture of the A7 read/write control logic. It is divided into two main sections: read control (top) and write control (bottom).

Read Control Section:

- user_rd_ctrl:** Receives external signals: `clk`, `rst_n`, `rd_start`, and `p1_rd_count`. It outputs `p1_rd_en`, `p1_rd_data`, `p1_rd_empty`, `p1_rd_full`, and `p1_rd_count` to the `rd data fifo`.
- rd data fifo:** Outputs `rd data` to the `ctrl` block.
- ctrl:** Receives `p1_cmd_en`, `p1_cmd_instr`, `p1_cmd_b1`, `p1_cmd_addr`, and `p1_cmd_full`. It outputs `rd_cmd_b1`, `rd_cmd_addr`, `rd_cmd_instr`, `rd_cmd_start`, and `rd_end` to the `A7_rd_ctrl` block.
- ARBIT:** Receives `rd_req` and `wr_req`. It outputs `rd_req` to the `rd data fifo` and `wr_req` to the `wr cmd fifo`.
- A7_rd_ctrl:** Receives `rd_data_valid`, `rd_data_128bit`, `rd_cmd_b1`, `rd_cmd_addr`, `rd_cmd_instr`, `rd_cmd_start`, and `rd_end`. It outputs `app_rd_data`, `app_rd_data_valid`, `app_rd_data_end`, `app_rdy`, `app_addr`, `app_en`, and `app_cmd` to the `IP` block.

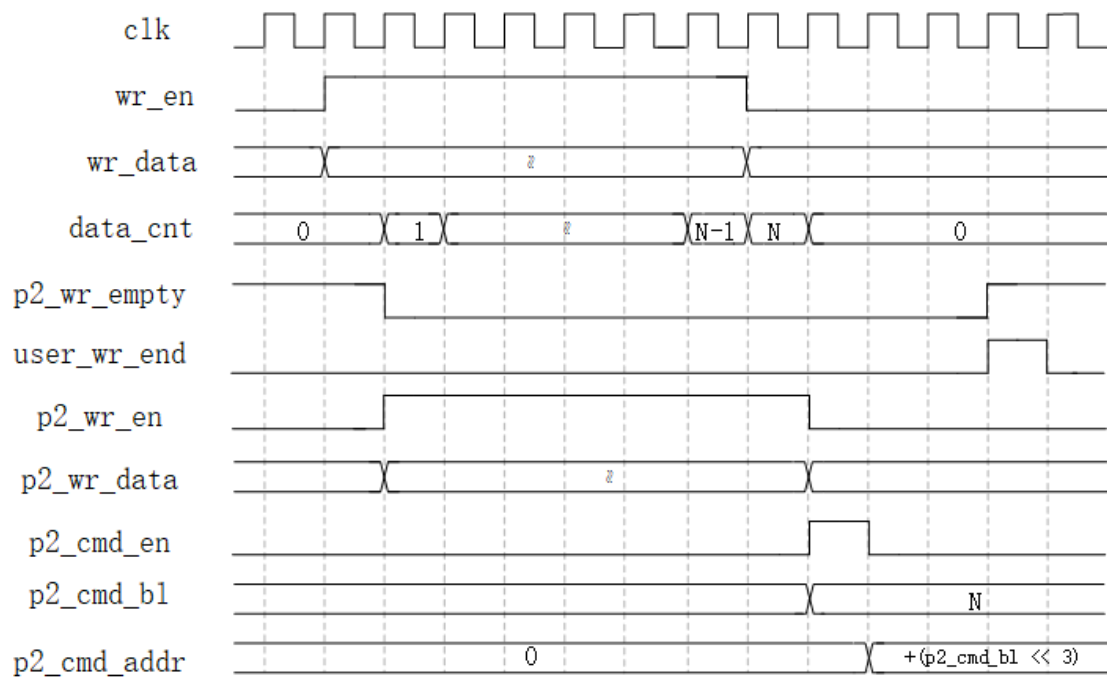
Write Control Section:

- user_wr_ctrl:** Receives external signals: `clk`, `rst_n`, `wr_en`, `wr_data`, and `p2_wr_empty`. It outputs `p2_cmd_en`, `p2_cmd_instr`, `p2_cmd_b1`, `p2_cmd_addr`, `p2_cmd_full`, `p2_wr_en`, `p2_wr_data`, `p2_wr_mask`, `p2_wr_empty`, `p2_wr_full`, and `p2_wr_count` to the `wr cmd fifo` and `wr data fifo`.
- wr cmd fifo:** Outputs `wr_cmd_b1`, `wr_cmd_addr`, `wr_cmd_instr`, and `wr_cmd_full` to the `ctrl` block.
- ctrl:** Receives `wr_cmd_b1`, `wr_cmd_addr`, `wr_cmd_instr`, and `wr_cmd_full`. It outputs `wr_cmd_b1`, `wr_cmd_addr`, `wr_cmd_instr`, `wr_cmd_mask`, `data_128bit`, and `data_req` to the `A7_wr_ctrl` block.
- wr data fifo:** Outputs `wr data` to the `ctrl` block.
- A7_wr_ctrl:** Receives `ui_clk`, `init_calib_data`, `wr_cmd_b1`, `wr_cmd_addr`, `wr_cmd_instr`, `wr_cmd_mask`, `data_128bit`, and `data_req`. It outputs `app_cmd`, `app_en`, `app_addr`, `app_rdy`, `app_vdf_data`, `app_vdf_wren`, `app_vdf_mask`, `app_vdf_end`, and `app_vdf_rdy` to the `IP` block.

The `IP` block is the final destination for the data and control signals from the A7 control logic.

1. 写时序分析

- 数据FIFO写入完毕后，从数据FIFO中读取数据，即此时开始执行写入数据的命令，这部分的控制采用wr_en来把控；
- 数据FIFO数据完毕后，一帧数据读取完毕，拉起写完成标志，即wr_end拉高；



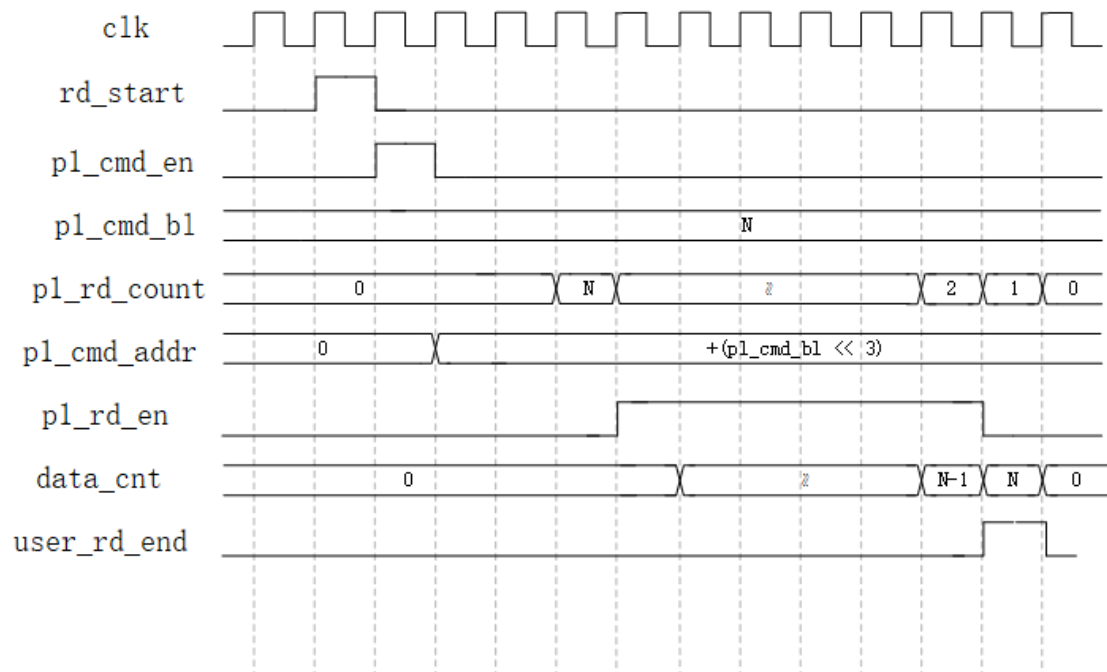
写时序控制

2. 读时序分析

(1) 关键点时序

- rd_start后，立即写入读取数据的命令，命令中cmd_b1是固定的；
- 等待数据FIFO中存储的数据和要求读取的数据一致时，开始读取数据，即pl_rd_en拉高；

(2) 时序波形



读时序控制

3. 仿真实现分析

(1) 写数据

写完64突发长度的数据后，需等待wr_end拉高后，再次写入64突发长度的数据。

(2) 读数据

读完64突发长度的数据后，需等待rd_end拉高后，再次读取64突发长度的数据。

四、练习步骤

1. 按照分析的时序编写代码
 2. 按照仿真分析编写测试代码
- (1) 写仿真

```
task gen_wr_data;
    integer i;
    begin
        @ (negedge rst);
        repeat (100) @ (posedge clk);
        wr_en = 1;
        for (i = 0; i < 64; i = i + 1)
            begin
                wr_data = i;
                @ (posedge clk);
            end
        wr_en = 0;
        @ (posedge clk);
        @ (negedge user_wr_end);
        repeat (100) @ (posedge clk);
        wr_en = 1;
        for (i = 0; i < 64; i = i + 1)
            begin
                wr_data = i + 64;
                @ (posedge clk);
            end
        wr_en = 0;
    end
endtask
```

(2) 读仿真

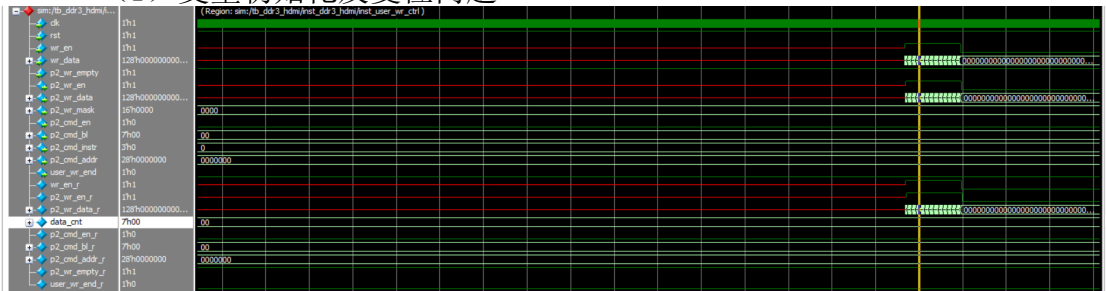
```
task gen_rd_data;
begin
    @ (negedge rst);
    @ (negedge user_wr_end);
    @ (negedge user_wr_end);    //等待写入完毕
    rd_start = 1;
    @ (posedge clk);
    rd_start = 0;
    @ (posedge clk);
    @ (negedge user_rd_end);    //等待读取完毕1
    rd_start = 1;
    @ (posedge clk);
    rd_start = 0;
    @ (posedge clk);
end
endtask
```

3. 查看仿真结果
- (1) 对比时序是否正确
 - (2) 对比数据是否正常

五、实际波形仿真

1. 写时序仿真测试

(1) 变量初始化及复位问题

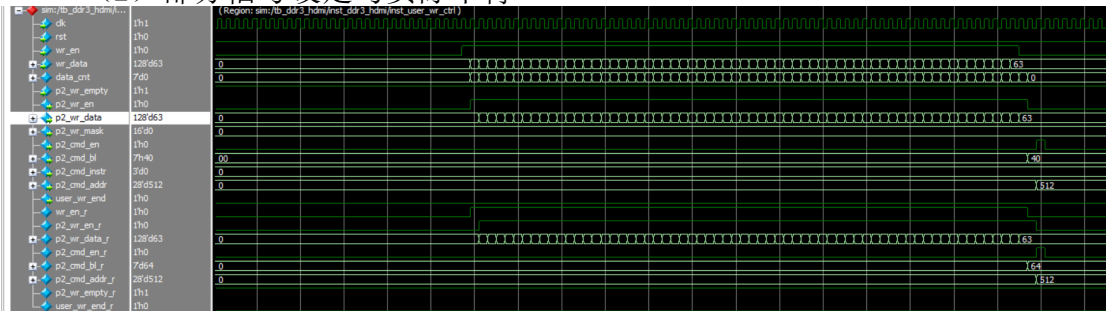


变量未正确的初始化，即某些变量并未发生改变，一致保持不变，这就是复位出现了错误，导致一致保持为零，这里需要进行简单的修改以达到要求。

```
85  
86  
87 wr_en = 0;  
88 wr_data = 0;  
rd_start = 0;
```

首先，在testbench文件中添加相应的变量初始化代码；
然后，修改为高电平的复位已达到要求。
再重新查看仿真结果。

(2) 部分信号设定与实际不符



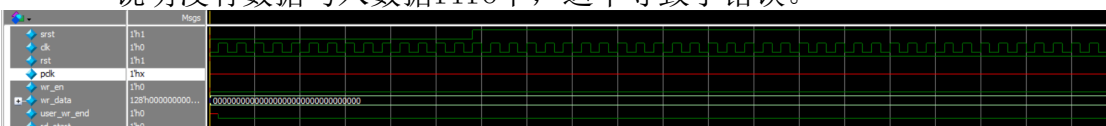
- `p2_cmd_en`与设计时序不符

```
86 //p2_cmd_en  
87 always @(posedge clk)  
88 begin  
89     if (rst == 1'b1)  
90         p2_cmd_en_r <= 1'b0;  
91     else if (wr_en == 1'b0 && wr_en_r == 1'b1) //下降沿检测  
92         p2_cmd_en_r <= 1'b1;  
93     else  
94         p2_cmd_en_r <= 1'b0;  
95 end
```

发现`p2_cmd_en`波形与实际设计波形相比，滞后了一个时钟周期，这里将其修改为上述代码，即可达标。

- 数据fifo一直为空的

说明没有数据写入数据fifo中，这个导致了错误。



经检查，是时钟信号出现了问题，一直没有时钟信号，故没有数据输入。

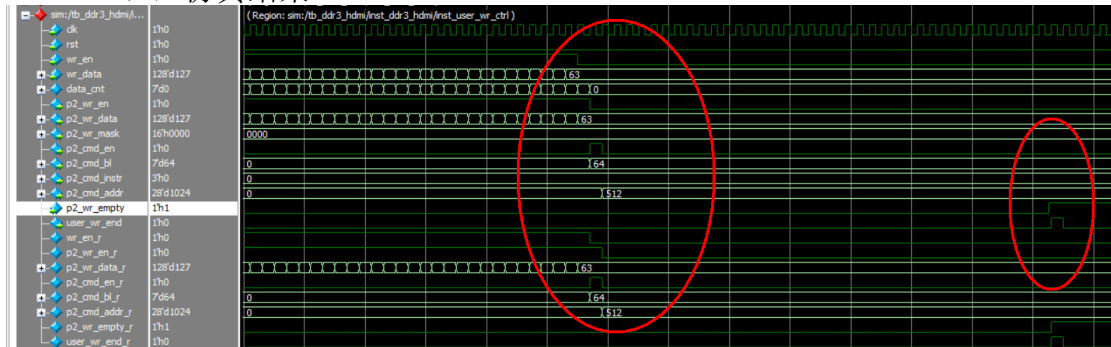
```

129 //fifo clock
130 initial begin
131     pclk = 0;
132     forever #(10) pclk = ~pclk;
133 end

```

发现是时钟信号没有初始化，增加初始化代码，再次进行仿真。

(3) 仿真结果



实际波形与仿真波形完全一致，这部分时序基本上没有什么问题。

```

# tb_ddr3_hdm1.inst_ddr3_model.main: at time 115118314.0 ps INFO: Sync On Die Termination Rtt_NGM = 60 Ohm
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115127064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003f8 data = 007f
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115128314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003f9 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115129564.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003fa data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115130814.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003fb data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115132064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003fc data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115133314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003fd data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115134564.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003fe data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115135814.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 000003ff data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110267064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000000 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110268314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000001 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110269564.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000002 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110270814.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000003 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110272064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000004 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110273314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000005 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110273314.0 ps INFO: Write bank 0 col 010, auto precharge 0
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110274564.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000006 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110275814.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000007 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110277064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000008 data = 0001
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110278314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 00000009 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110279564.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 0000000a data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110280814.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 0000000b data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110282064.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 0000000c data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 110283314.0 ps INFO: WRITE @ DQS= bank = 0 row = 0000 col = 0000000d data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110283314.0 ps INFO: Write bank 0 col 018, auto precharge 0

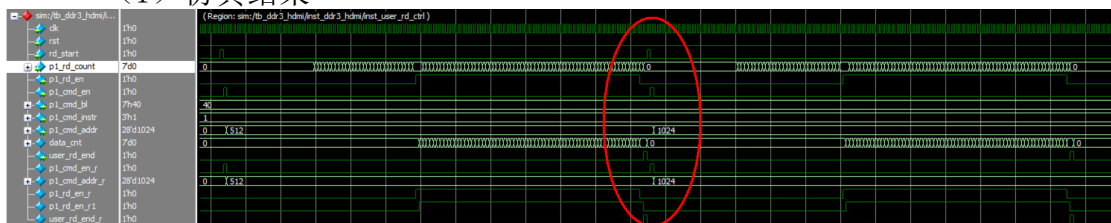
```

实际写入数据也是如此，这样，写时序部分，就没有什么问题啦。

2. 读时序仿真测试

这部分，没什么问题，可以直接查看测试结果。

(1) 仿真结果



波形与时序分析时一致，程序实现数据读取。

```

# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115167064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000000 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115168314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000001 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115169564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000002 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115170814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000003 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115172064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000004 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115173314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000005 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 115173314.0 ps INFO: Read bank 0 col 010, auto precharge 0
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115174564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000006 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115175814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000007 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115177064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000008 data = 0001
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115178314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 00000009 data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115179564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 0000000a data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115180814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 0000000b data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115182064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 0000000c data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.data_task: at time 115183314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 0000000d data = 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 115183314.0 ps INFO: Read bank 0 col 018, auto precharge 0

```

```

# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118215814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003ef data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118217064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f0 data = 007e
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118218314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f1 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118219564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f2 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118220814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f3 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118222064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f4 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118223314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f5 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118224564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f6 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118225814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f7 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118227064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f8 data = 007f
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118228314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003f9 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118229564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003fa data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118230814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003fb data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118232064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003fc data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118233314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003fd data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118234564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003fe data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 118235814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000003ff data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.cmd_task: at time 118255814.0 ps INFO: Precharge bank 0

```

读取数据与写入数据一致，总的来说，成功。

六、总结与讨论

1. 仿真时长为120us
2. 尽可能简化的话，有些变量是必须是常量的，这是不可避免的。
3. 关于代码实现的几点注意

(1) addr的清零问题

DDR3最终是用于HDMI显示的，考虑HDMI显示的数据量，当一帧数据写入和读取完毕后，即一幅图像的数据传输完毕后，需要将地址清零。

- 考虑的分辨率是1024*768，固定突发长度是64，即每次最大能够写入64*128bit的数据，对于16bit的图像数据来说， $64*128/16 = 512$ 个数据，传输完一帧数据则需要768*2=1536次

```

28 //1024*768
29 //16bit / pixel
30 //1024*768 - 512 = 785920
31 parameter BURST_LEN = 64;
32 parameter START_ADDR = 0;
33 parameter STOP_ADDR = 785920;
34 parameter ADD_ADDR = 64*128/16; //=512

```

- 对于地址来说，地址到达1024*768-512后，就需要清零，因为在这一帧数据传输完毕。

```

76 always @(posedge clk)
77 begin
78     if (rst == 1'b1)
79         wr_cmd_addr_r <= START_ADDR;
80     else if (wr_cmd_addr_r == STOP_ADDR && wr_cmd_en_r == 1'b1)
81         wr_cmd_addr_r <= START_ADDR;
82     else if (wr_cmd_en_r == 1'b1)
83         wr_cmd_addr_r <= wr_cmd_addr_r + ADD_ADDR;
84 end

```

(2) user_wr_ctrl中p2_wr_empty信号实现的不一致性问题解决。

- 追根溯源，我们发现p2_wr_empty使用的时钟是ui_clk，而user_wr_ctrl使用的时钟为p2_clk;


```

194 wr_data_fifo_ctrl wr_data_fifo_inst (
195     .rst(ui_clk_sync_rst | (~init_calib_complete)),
196     .wr_clk(p2_clk),           // input wire wr_clk
197     .rd_clk(ui_clk),           // input wire rd_clk
198     .din({p2_wr_mask, p2_wr_data}), // input wire
199     .wr_en(p2_wr_en),           // input wire wr_en
200     .rd_en(data_req),           // input wire rd_en
201     .dout({wr_cmd_mask, data_128bit}), // output wire
202     .full(p2_wr_data_full),     // output wire
203     .empty(p2_wr_data_empty),   // output wire
204     .wr_data_count(p2_wr_data_count), // output wire [6 : 0]
205     .wr_rst_busy(),             // output wire wr_rst_busy
206     .rd_rst_busy()             // output wire rd_rst_busy
207 );

```

- 两者时钟的不一致性，会导致亚稳态，这里采用延时俩拍的方式解决；

```

85
86 //亚稳态，打俩拍消除
87 always @(posedge clk)
88 begin
89     wr_empty_r <= p2_wr_empty;
90     wr_empty_rr <= wr_empty_r;
91     wr_empty_rrr <= wr_empty_rr;
92 end

```

(3) user_rd_ctrl中user_rd_end的拉高方式优化

- 可以采用p1_rd_en延时一拍，提取下降沿的方式作为采样信号；

```

92 //user_rd_end
93 always @(posedge clk)
94 begin
95     if (rst == 1'b1)
96         user_rd_end_r <= 1'b0;
97     else if (p1_rd_en == 1'b0 && p1_rd_en_r1 == 1'b1)
98         user_rd_end_r <= 1'b1;
99     else
100         user_rd_end_r <= 1'b0;
101 end

```

- 更好的方式是 data_cnt == BURST_LEN 作为判别方式；

```

86 always @(posedge clk)
87 begin
88     if (rst == 1'b1)
89         rd_end_r <= 1'b0;
90     else if (data_cnt == (BURST_LEN - 1))
91         rd_end_r <= 1'b1;
92     else
93         rd_end_r <= 1'b0;
94 end

```

总的来说，我们跟老师相比，还有很多需要学，也值得我们去学。

