

高级练习4 A7 的 DDR3 IP 核外围添加 FIFO 接口控制器的实现

笔记本: FPGA练习

创建时间: 2019/11/13 15:23

更新时间: 2020/2/3 16:41

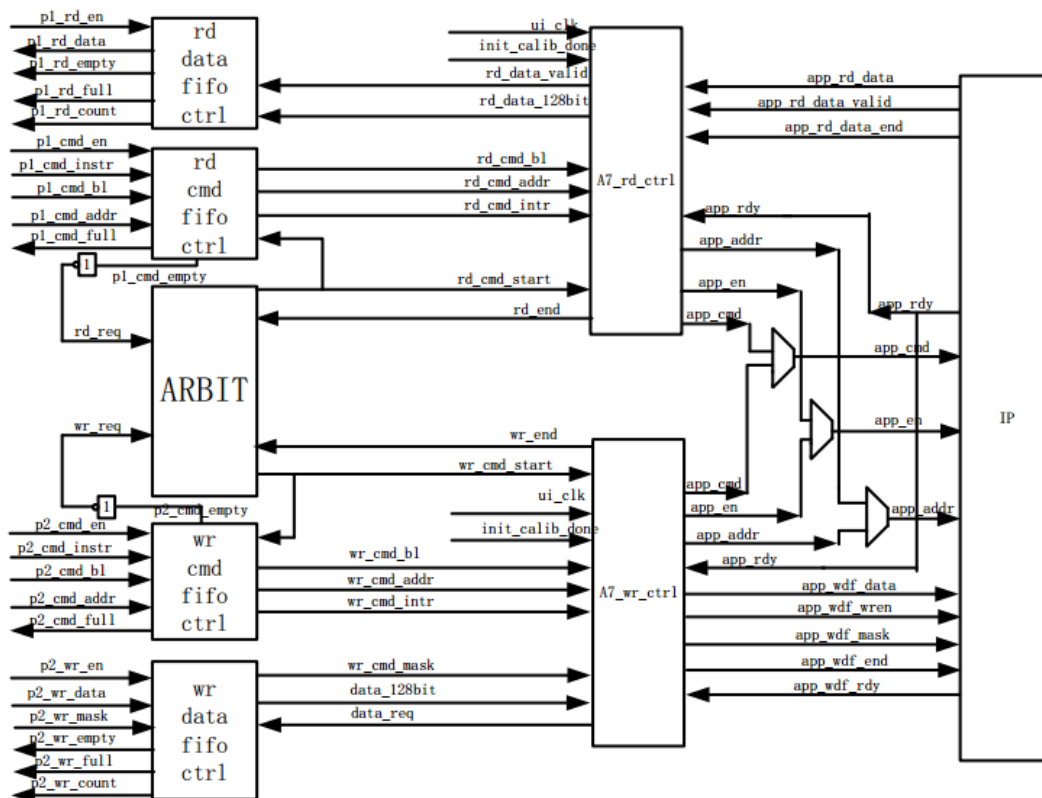
作者: 2322900041@qq.com

高级练习4 A7 的 DDR3 IP 核外围添加 FIFO 接口控制器的实现

一、练习内容

A7 的 DDR3 IP 核外围添加 FIFO 接口控制器的实现

二、系统框图



三、设计分析

1. 前面内容回顾

(1) ddr3写控制

- 先发送数据，再发送命令
- rdy和en都有效时，写入的数据和命令有效
- 在此DDR3写入数据的控制信号中，app_wdf_wren和app_wdf_end的信号是一致的

(2) ddr3读控制

- 在此DDR3读取数据的控制信号中，app_rd_data_end和app_rd_data_valid信号是一致的
- 先发送命令，再读取数据
- app_rdy和app_en都有效时，写入的命令有效

(3) ddr3读写仲裁 (*)

- 前提：读写不同步
- 读写使能信号不同步：app_wr_en和app_rd_en直接相或实现
- 读写命令不能同时存在：app_cmd在写命令有效时，即app_wr_en为1时，app_cmd的数据从app_wr_cmd输入，否则从app_rd_cmd输入
- 读写地址在读写完成后，直接清零，保证读写不同步：app_wr_addr和app_rd_addr直接相或实现
- 读写共用rdy信号：app_rdy连接至app_wr_rdy和app_rd_rdy

2. 读写命令fifo控制信号分析

(1) 写命令fifo

- 写命令fifo非空时，empty信号不为零，取反作为ARBIT的写请求wr_req的控制信号
- ARBIT的wr_cmd_start信号作为写命令fifo的读使能rd_en信号

(2) 读命令fifo

- 读命令fifo非空时，empty信号不为零，取反作为ARBIT的读请求rd_req的控制信号
- ARBIT的rd_cmd_start信号作为读命令fifo的读使能rd_en信号

3. 读写数据fifo控制信号分析

(1) 写数据fifo：ddr3_wr_ctrl写入的数据从写数据fifo中获取

- data_req作为写数据fifo的读使能
- data_128bit作为写数据fifo的写数据

(2) 读数据fifo：ddr3_rd_ctrl读取的数据存储在读数据fifo中

- rd_data_valid作为读数据fifo的写使能
- rd_data_128bit作为读数据fifo的写数据

四、练习步骤

1. verilog程序编写

这里主要涉及到四个模块程序的编写，分别是wr_data_fifo_ctrl、wr_cmd_fifo_ctrl、rd_cmd_fifo_ctrl和rd_data_fifo_ctrl的fifo控制，这里分别进行介绍。

(1) wr_data_fifo_ctrl

- 首先是fifo模块的设置
 - 数据位数达到144位，深度达到64个，这个和wr_cmd_bl的最大达到64有关。
 - 然后采用的First Word Fall Through模式
 - 输出写入的具体数据长度

Basic	Native Ports	Status Flags	Data Counts	Summary
Read Mode <input type="radio"/> Standard FIFO <input checked="" type="radio"/> First Word Fall Through				
Data Port Parameters				
Write Width	144	✕	1,2,3,...1024	
Write Depth	64	▼	Actual Write Depth: 65	
Read Width	144	▼		
Read Depth	64		Actual Read Depth: 65	

- 然后是部分控制信号的控制

对于具体的控制信号，参考功能框图即可，对于这部分的说明是很详细的。

(2) wr_cmd_fifo_ctrl

- 首先是fifo模块的设置
 - 数据位数达到38位，深度达到16个，设置成16已经够用了。
 - 然后采用的First Word Fall Through模式

Basic	Native Ports	Status Flags	Data Counts	Summary
Read Mode <input type="radio"/> Standard FIFO <input checked="" type="radio"/> First Word Fall Through				
Data Port Parameters				
Write Width	38	✕	1,2,3,...1024	
Write Depth	16	▼	Actual Write Depth: 17	
Read Width	38	▼		
Read Depth	16		Actual Read Depth: 17	

- 然后是部分控制信号的控制

对于具体的控制信号，参考功能框图即可，对于这部分的说明是很详细的。

(3) rd_cmd_fifo_ctrl

- 首先是fifo模块的设置，和wr_cmd_fifo_ctrl是一致的
 - 数据位数达到38位，深度达到16个，设置成16已经够用了。
 - 然后采用的First Word Fall Through模式

Basic	Native Ports	Status Flags	Data Counts	Summary
Read Mode <input type="radio"/> Standard FIFO <input checked="" type="radio"/> First Word Fall Through				
Data Port Parameters				
Write Width	38	⊗	1,2,3,...1024	
Write Depth	16	▼	Actual Write Depth: 17	
Read Width	38	▼		
Read Depth	16		Actual Read Depth: 17	

- 然后是部分控制信号的控制

对于具体的控制信号，参考功能框图即可，对于这部分的说明是很详细的。

(4) rd_data_fifo_ctrl

- 首先是fifo模块的设置
 - 数据位数达到144位，深度达到64个，这个和wr_cmd_b1的最大达到64有关。
 - 然后采用的First Word Fall Through模式

Basic	Native Ports	Status Flags	Data Counts	Summary
Read Mode <input type="radio"/> Standard FIFO <input checked="" type="radio"/> First Word Fall Through				
Data Port Parameters				
Write Width	128	⊗	1,2,3,...1024	
Write Depth	64	▼	Actual Write Depth: 65	
Read Width	128	▼		
Read Depth	64		Actual Read Depth: 65	

- 然后是部分控制信号的控制

对于具体的控制信号，参考功能框图即可，对于这部分的说明是很详细的。

2. testbench程序编写

对于testbench的编写也是如此，这里需要注意的是整个fifo执行的先后顺序。

(1) 先写入数据

写使能的控制是以ddr3初始化完成后开始的，即@ (negedge rst);

然后连续写入64个数据，这个主要注意的。

(2) 再写入写数据的命令

主要涉及到wr_cmd_b1设为64，wr_cmd_addr设为0，wr_cmd_instr设置为0；

然后需要注意的执行的先后顺序，需要在写入数据后，立马写入地址，这里采用的方式是开始写入数据使能后，立马写入写数据命令，即@ (posedge p2_wr_en);

(3) 然后写入读数据的命令

主要涉及到rd_cmd_b1设为64, rd_cmd_addr设为0,
rd_cmd_instr设置为1;

然后需要注意的执行的先后顺序, 需要在写入数据后, 再考虑读取数据, 这里采用的方式是数据写入完毕后, 立马写入读数据的命令, 即@
(negedge p2_wr_en);

(4) 最后读取数据

考虑写入读取数据的命令后, 再读取数据, 即@
posedge(p1_cmd_en), 再考虑将p1_rd_en拉高。

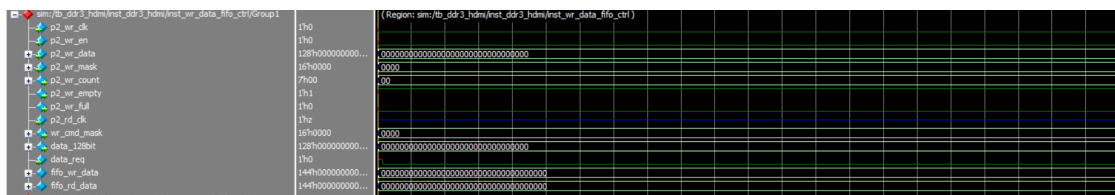
五、实际波形仿真

1. 局部查看信号

加入上述4个模块, 同时加入ddr3_wr_ctrl和ddr3_rd_ctrl模块, 一步步查看数据是否正确写入和读取。

(1) 数据写入

- 时钟信号存在问题



查看波形, 发现没有时钟信号输入, 后面的过程根本无法执行。

```
243 task gen_clk(clk_in);
244 begin
245     forever #(10) clk_in = ~clk_in;
246 end
247 endtask
```

需要修改代码, 再考虑问题。

```
252 task gen clk;
253     input clk_in;
254     begin
255         forever #(10) clk_in = ~clk_in;
256     end
257 endtask
```

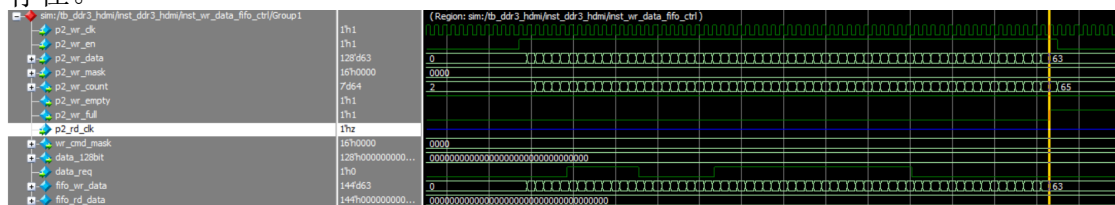
如上图所示。

最终改成下面, 这种形式, 可以得到正确的时钟信号。

```
164 //fifo clock
165 always #10 p2_wr_clk = ~ p2_wr_clk;
166 always #10 p2_cmd_clk = ~ p2_cmd_clk;
167 always #10 p1_cmd_clk = ~ p1_cmd_clk;
168 always #10 p1_rd_clk = ~ p1_rd_clk;
```

- 数据读取存在问题

数据写入没有什么问题, 所有数据基本正确的写入, 读取时钟rd_clk不存在。



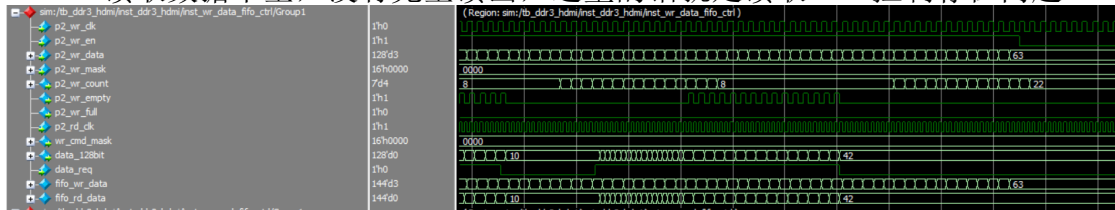
发现是调用时时钟出现错误, 这里进行修正,

```

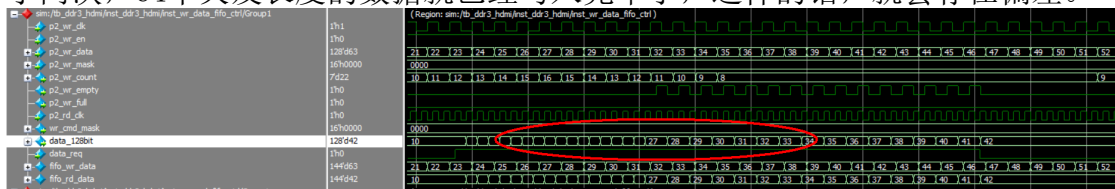
232 wr_data_fifo_ctrl inst_wr_data_fifo_ctrl
233 (
234     .p2_wr_clk      (p2_wr_clk),
235     .p2_wr_en       (p2_wr_en),
236     .p2_wr_data     (p2_wr_data),
237     .p2_wr_mask     (p2_wr_mask),
238     .p2_wr_count    (p2_wr_count),
239     .p2_wr_empty    (p2_wr_empty),
240     .p2_wr_full     (p2_wr_full),
241     .p2_rd_clk      (ui_clk),
242     .wr_cmd_mask    (wr_cmd_mask),
243     .data_128bit    (data_128bit),
244     .data_req       (data_req)
245 );

```

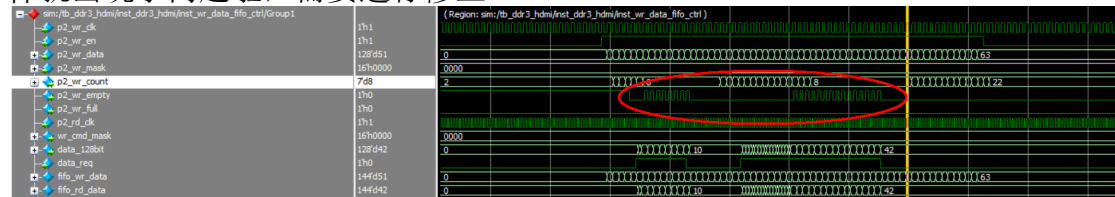
读取数据不全，没有完全读出，这里的话就是读取fifo控制存在问题。



通过观察波形发现，部分fifo的读取数据的占用宽度不一致，导致写入了两次，64个突发长度的数据就已经写入完毕了，这样的话，就会存在偏差。



再次观察波形，发现读取数据的过程中，存在多次fifo为空的情形，这样就出现了问题啦，需要进行修正。



重新修改读取数据的条件为fifo写满，保证数据读取不会空读，出现读不到数据的情况，避免出现了上述问题。

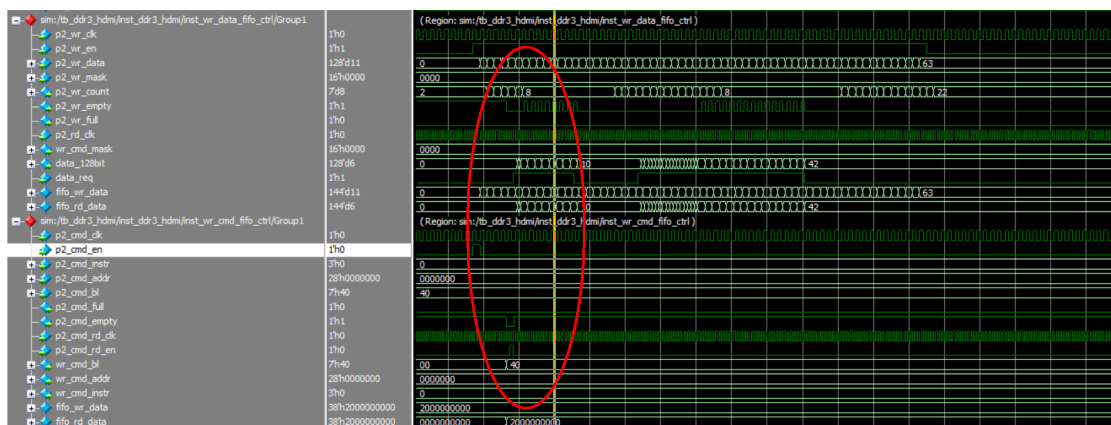
```

205 task gen_rd_cmd;
206 begin
207     @ (negedge rst);
208     @(posedge p2_wr_full);
209     p1_cmd_en = 1;
210     @ (posedge p1_cmd_clk);
211     p1_cmd_en = 0;
212 end
213 endtask

```

发现不是这个方向的错误，这个问题仍然存在。

重新观察，发现是写入数据命令操作过快，写入数据后，并未完全写入fifo中，这完全是因为读取时序过快的原因。这里需要等待数据完全写入后，再进行相关操作。



修改如下，再重新进行操作。保证数据的正确性和准确性。

```

205 task gen_rd_cmd;
206 begin
207     @(negedge rst);
208     @(posedge p2_wr_empty); //等待写入数据fifo为空，即数据全部写入ddr3中
209     p1_cmd_en = 1;
210     @(posedge p1_cmd_clk);
211     p1_cmd_en = 0;
212 end
213 endtask
214
215 task gen_wr_cmd;
216 begin
217     @(negedge rst);
218     @(negedge p2_wr_en); //等待数据完全写入fifo中
219     p2_cmd_en = 1;
220     @(posedge p2_cmd_clk);
221     p2_cmd_en = 0;
222 end
223 endtask

```

(2) 数据输出

```

Transcript
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109714564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001ee data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109715814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001ef data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109717064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f0 data = 003e
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109718314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f1 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109719564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f2 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109720814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f3 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109722064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f4 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109723314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f5 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109724564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f6 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109725814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f7 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109727064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f8 data = 003f
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109728314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001f9 data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109729564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001fa data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109730814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001fb data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109732064.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001fc data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109733314.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001fd data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109734564.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001fe data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.data_task: at time 109735814.0 ps INFO: READ @ DQS= bank = 0 row = 0000 col = 000001ff data = 0000
# tb_ddr3_hdmi.inst_ddr3_model.cmd_task: at time 109755814.0 ps INFO: Precharge bank = 0

```

数据输出结果正确，然后观察检测部分是否正确，是否报错。

发现一直没有提示输出，这里进行修改，保证可以进行比较，然后在查看结果。


```

186 task get_rd_data;
187     integer i;
188     begin
189         @ (negedge rst);
190         @ (negedge p1_rd_empty);           //等待fifo非空
191         p1_rd_en = 1;
192         for (i = 0; i < p1_cmd_bl; i = i + 1)
193             begin
194                 if (i == p1_rd_data)
195                     $display("read %2d right!", p1_rd_data);
196                 else
197                     $display("read %2d error!", p1_rd_data);
198                 @ (posedge p1_rd_clk);
199             end
200         @ (posedge p1_rd_empty);           //等待fifo为空
201         p1_rd_en = 0;
202     end
203 endtask

```

2. 整个数据输出把控
查看是否有出错提示

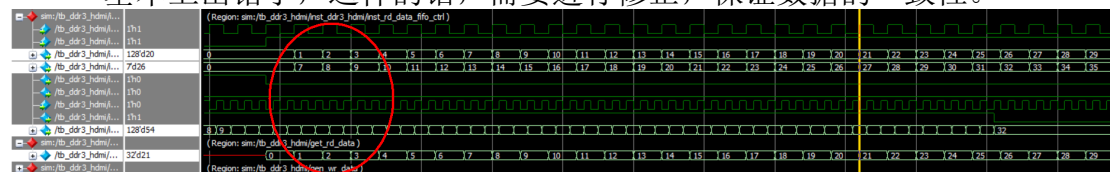
Transcript

```

# read 46 error!
# read 47 error!
# read 48 error!
# read 49 error!
# read 50 error!
# read 51 error!
# read 52 error!
# read 53 error!
# read 54 error!
# read 55 error!
# read 56 error!
# read 57 error!
# read 58 error!
# read 59 error!
# read 60 error!
# read 61 error!
# read 62 error!
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110630814.0 ps INFO: Activate bank 0 row 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110653314.0 ps INFO: Read bank 0 col 1f8, auto precharge 0

```

基本上出错了，这样的话，需要进行修正，保证数据的一致性。



重新查看结果为数据完全一致，没有任何问题。

Transcript

```

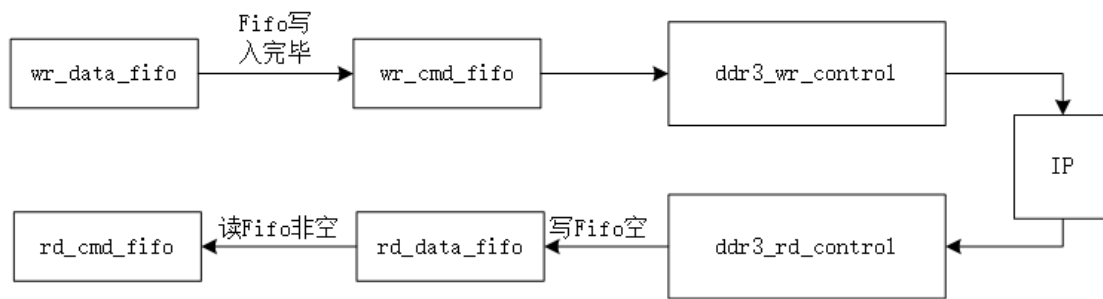
# read 47 right!
# read 48 right!
# read 49 right!
# read 50 right!
# read 51 right!
# read 52 right!
# read 53 right!
# read 54 right!
# read 55 right!
# read 56 right!
# read 57 right!
# read 58 right!
# read 59 right!
# read 60 right!
# read 61 right!
# read 62 right!
# read 63 right!
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110630814.0 ps INFO: Activate bank 0 row 0000
# tb_ddr3_hdm1.inst_ddr3_model.cmd_task: at time 110653314.0 ps INFO: Read bank 0 col 1f8, auto precharge 0

```

整个实验到此结束，搞定。

六、总结与讨论

1. 不一定要完全知道怎么做了，才去尝试，可以在不断的练习和修订中得到提升；
2. 有想法也得有行动才行啊，不然只会做无用功；
3. 关于整个执行的实现过程，绘制如下的示意框图。



4. 还有就是仿真的时候需要达到的时间为112us, 这是最少的, 太少了看不到所有的数据。