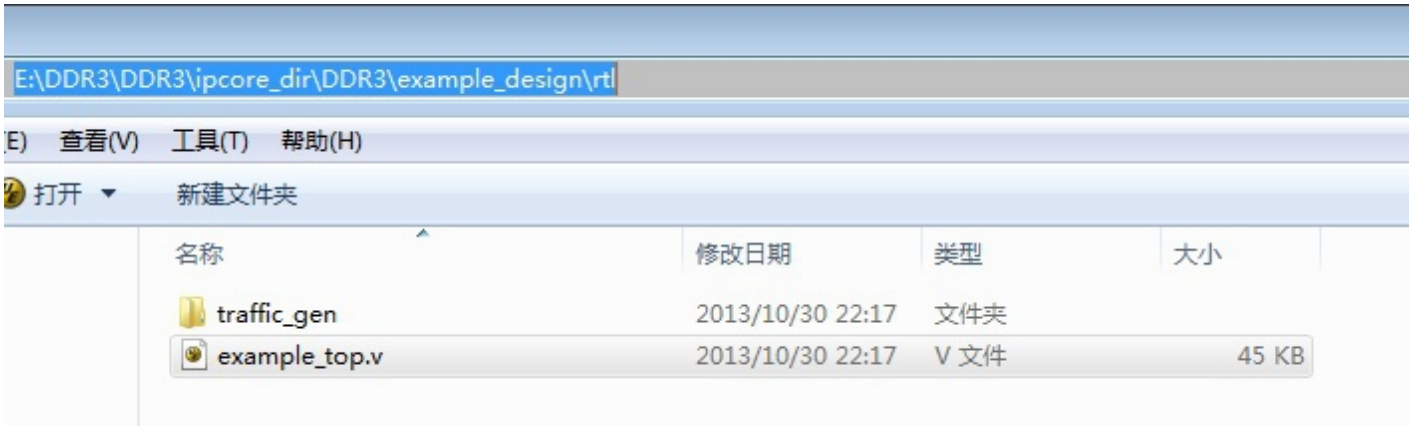
现在你应该已经看完了仿真和综合教程
我们进入了设计篇，说白了就是讲一讲DDR IP的用户接口是怎么用的

用户接口在哪里？

请你打开下面这个目录里面的example_top.v

这也就是你综合出来工程的顶层文件了



我们来理一理
这个文件的结构吧
开头部分，

全是介绍，
你删了都关系

CPU与内存之间的接口位宽是64bit，也就意味着CPU在一个时钟周期内会向内存发送或从内存读取64bit的数据。可是，单个内存颗粒的位宽仅有4bit、8bit或16bit，个别也有32bit的。因此，必须把多个颗粒并联起来，组成一个位宽为64bit的数据集合，才可以和CPU互连。生产商把64bit集合称为一个物理BANK（Physical BANK），简写为P-BANK。为了和逻辑BANK相区分，也经常把P-BANK称为RANK或Physical RANK,把L-BANK则简称为BANK。

```
1  //*****************************************************************************
2  // (c) Copyright 2009 - 2010 Xilinx, Inc. All rights reserved.
3  //
4  // This file contains confidential and proprietary information
5  // of Xilinx, Inc. and is protected under U.S. and
6  // international copyright and other intellectual property
7  // laws.
8  //
9  // DISCLAIMER
10 // This disclaimer is not a license and does not grant any
11 // rights to the materials distributed herewith. Except as
12 // otherwise provided in a valid license issued to you by
13 // Xilinx, and to the maximum extent permitted by applicable
14 // law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
15 // WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
16 // AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
17 // BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
18 // INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
19 // (2) Xilinx shall not be liable (whether in contract or tort,
20 // including negligence, or under any other theory of
21 // liability) for any loss or damage of any kind or nature
22 // related to, arising under or in connection with these
23 // materials, including for any direct, or any indirect,
24 // special, incidental, or consequential loss or damage
25 // (including loss of data, profits, goodwill, or any type of
26 // loss or damage suffered as a result of any action brought
27 // by a third party) even if such damage or loss was
28 // reasonably foreseeable or Xilinx had been advised of the
29 // possibility of the same.
30 //
31 // CRITICAL APPLICATIONS
32 // Xilinx products are not designed or intended to be fail-
33 // safe, or for use in any application requiring fail-safe
34 // performance, such as life-support or safety devices or
35 // systems, Class III medical devices, nuclear facilities,
36 // applications related to the deployment of airbags, or any
37 // other applications that could lead to death, personal
38 // injury, or severe property or environmental damage
39 // (individually and collectively, "Critical
40 // Applications"). Customer assumes the sole risk and
41 // liability of any use of Xilinx products in Critical
42 // Applications, subject only to applicable laws and
43 // regulations governing limitations on product liability.
44 //
45 // THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
46 // PART OF THIS FILE AT ALL TIMES.
47 //
48 //*****************************************************************************
49 //
```

然后是各种参数的设定

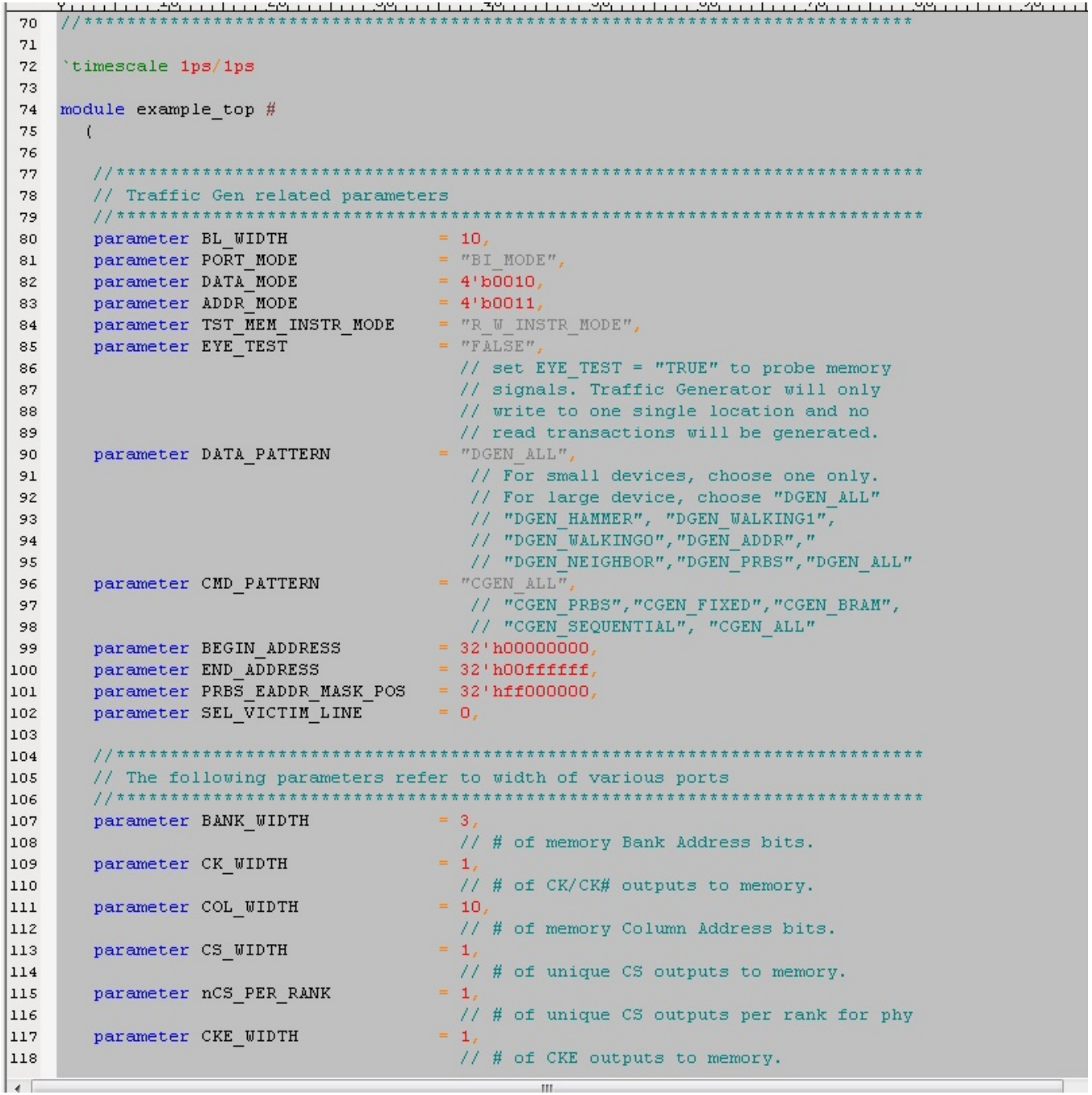
这里有bank，row，column，rank，等等各种设置

其实你不用动它们

这些都是你之前选条子的时候已经选好了的

不记得自己选什么条子了？

乖乖，你不如再翻翻仿真教程先？



```
70  //**************************************************************************
71
72  `timescale 1ps/1ps
73
74  module example_top #
75    (
76
77     //***************************************************************************
78     // Traffic Gen related parameters
79     //***************************************************************************
80     parameter BL_WIDTH              = 10,
81     parameter PORT_MODE             = "BI_MODE",
82     parameter DATA_MODE             = 4'b0010,
83     parameter ADDR_MODE             = 4'b0011,
84     parameter TST_MEM_INSTR_MODE    = "R_W_INSTR_MODE",
85     parameter EYE_TEST              = "FALSE",
86                                       // set EYE_TEST = "TRUE" to probe memory
87                                       // signals. Traffic Generator will only
88                                       // write to one single location and no
89                                       // read transactions will be generated.
90     parameter DATA_PATTERN          = "DGEN_ALL",
91                                        // For small devices, choose one only.
92                                        // For large device, choose "DGEN_ALL"
93                                        // "DGEN_HAMMER", "DGEN_WALKING1",
94                                        // "DGEN_WALKING0","DGEN_ADDR","
95                                        // "DGEN_NEIGHBOR","DGEN_PRBS","DGEN_ALL"
96     parameter CMD_PATTERN           = "CGEN_ALL",
97                                        // "CGEN_PRBS","CGEN_FIXED","CGEN_BRAM",
98                                        // "CGEN_SEQUENTIAL", "CGEN_ALL"
99     parameter BEGIN_ADDRESS         = 32'h00000000,
100    parameter END_ADDRESS           = 32'h00ffffff,
101    parameter PRBS_EADDR_MASK_POS   = 32'hff000000,
102    parameter SEL_VICTIM_LINE       = 0,
103
104    //***************************************************************************
105    // The following parameters refer to width of various ports
106    //***************************************************************************
107    parameter BANK_WIDTH            = 3,
108                                      // # of memory Bank Address bits.
109    parameter CK_WIDTH              = 1,
110                                      // # of CK/CK# outputs to memory.
111    parameter COL_WIDTH             = 10,
112                                      // # of memory Column Address bits.
113    parameter CS_WIDTH              = 1,
114                                      // # of unique CS outputs to memory.
115    parameter nCS_PER_RANK          = 1,
116                                      // # of unique CS outputs per rank for phy
117    parameter CKE_WIDTH             = 1,
118                                      // # of CKE outputs to memory.
```

各种仿真延迟参数

也跟你选的条子有关

你也别管了

我都不管这些

```
238     // Memory Timing Parameters. These parameters varies based on the selected
239     // memory part.
240     //***************************************************************************
241     parameter tCKE                  = 5625,
242                                       // memory tCKE paramter in pS
243     parameter tFAW                  = 30000,
244                                       // memory tRAW paramter in pS.
245     parameter tRAS                  = 36000,
246                                       // memory tRAS paramter in pS.
247     parameter tRCD                  = 13500,
248                                       // memory tRCD paramter in pS.
249     parameter tREFI                 = 7800000,
250                                       // memory tREFI paramter in pS.
251     parameter tRFC                  = 160000,
252                                       // memory tRFC paramter in pS.
253     parameter tRP                   = 13500,
254                                       // memory tRP paramter in pS.
255     parameter tRRD                  = 6000,
256                                       // memory tRRD paramter in pS.
257     parameter tRTP                  = 7500,
258                                       // memory tRTP paramter in pS.
259     parameter tWTR                  = 7500,
260                                       // memory tWTR paramter in pS.
261     parameter tZQI                  = 128_000_000,
262                                       // memory tZQI paramter in nS.
263     parameter tZQCS                 = 64,
264                                       // memory tZQCS paramter in clock cycles.
265
266     //***************************************************************************
267     // Simulation parameters
268     //***************************************************************************
269     parameter SIM_BYPASS_INIT_CAL   = "OFF",
270                                       // # = "OFF" -  Complete memory init &
271                                       //              calibration sequence
272                                       // # = "SKIP" - Not supported
273                                       // # = "FAST" - Complete memory init & use
274                                       //              abbreviated calib sequence
275
276     parameter SIMULATION            = "FALSE",
277                                       // Should be TRUE during design simulations and
278                                       // FALSE during implementations
279
280     //***************************************************************************
281     // The following parameters varies based on the pin out entered in MIG GUI.
282     // Do not change any of these parameters directly by editing the RTL.
283     // Any changes required should be done through GUI and the design regenerated.
284     //***************************************************************************
285     parameter BYTE_LANES_B0         = 4'b1111,
286                                       // Byte lanes used in an IO column.
```

和DDR条子的各种接口

你要知道，用户接口是个内部接口，你这里当然看不到了。

如果之前选了
“use system clock”的话
这里就看不到
clk_ref相关的参考时钟管脚了。

这里顺便提一下
column和row地址是在ddr3_addr里面复用的。

column一般是10bit宽度。
row一般14-16bit宽度。
ddr3_ba是选bank的，
一般是3bit宽度，对应8个bank。
ddr3_cs_n是选rank的，
有几个rank就有几个bit的宽度，因为要考虑啥都不选的情况，和之前几个参数不一样的。

```
436   //***************************************************************************
437   // Debug parameters
438   //***************************************************************************
439   parameter DEBUG_PORT            = "OFF",
440                                     // # = "ON" Enable debug signals/controls.
441                                     //   = "OFF" Disable debug signals/controls.
442
443   parameter RST_ACT_LOW           = 1
444                                     // =1 for active low reset,
445                                     // =0 for active high.
446   )
447   (
448
449   // Inouts
450   inout [DQ_WIDTH-1:0]                        ddr3_dq,
451   inout [DQS_WIDTH-1:0]                       ddr3_dqs_n,
452   inout [DQS_WIDTH-1:0]                       ddr3_dqs_p,
453
454   // Outputs
455   output [ROW_WIDTH-1:0]                      ddr3_addr,
456   output [BANK_WIDTH-1:0]                     ddr3_ba,
457   output                                      ddr3_ras_n,
458   output                                      ddr3_cas_n,
459   output                                      ddr3_we_n,
460   output                                      ddr3_reset_n,
461   output [CK_WIDTH-1:0]                       ddr3_ck_p,
462   output [CK_WIDTH-1:0]                       ddr3_ck_n,
463   output [CKE_WIDTH-1:0]                      ddr3_cke,
464   output [CS_WIDTH*nCS_PER_RANK-1:0]          ddr3_cs_n,
465   output [DM_WIDTH-1:0]                       ddr3_dm,
466   output [ODT_WIDTH-1:0]                      ddr3_odt,
467
468   // Inputs
469   // Differential system clocks
470   input                                       sys_clk_p,
471   input                                       sys_clk_n,
472   // differential iodelayctrl clk (reference clock)
473   input                                       clk_ref_p,
474   input                                       clk_ref_n,
475
476   output                                      tg_compare_error,
477   output                                      init_calib_complete,
478
479
480   // System reset
481   input                                       sys_rst
482   );
483
```

各种参数配置
相互之间的关系换算，
选择

继续和你没有关系

作为设计者的你，
可以继续无视这些部分

```
484  function integer clogb2 (input integer size);
485      begin
486        size = size - 1;
487        for (clogb2=1; size>1; clogb2=clogb2+1)
488          size = size >> 1;
489      end
490    endfunction // clogb2
491
492    function integer STR_TO_INT;
493      input [7:0] in;
494      begin
495        if(in == "8")
496          STR_TO_INT = 8;
497        else if(in == "4")
498          STR_TO_INT = 4;
499        else
500          STR_TO_INT = 0;
501      end
502    endfunction
503
504
505    localparam CMD_PIPE_PLUS1        = "ON";
506                                       // add pipeline stage between MC and PHY
507    localparam DATA_WIDTH            = 64;
508    localparam ECC_WIDTH = (ECC == "OFF")?
509                             0 : (DATA_WIDTH <= 4)?
510                              4 : (DATA_WIDTH <= 10)?
511                               5 : (DATA_WIDTH <= 26)?
512                                6 : (DATA_WIDTH <= 57)?
513                                 7 : (DATA_WIDTH <= 120)?
514                                  8 : (DATA_WIDTH <= 247)?
515                                   9 : 10;
516    localparam ECC_TEST              = "OFF";
517    localparam RANK_WIDTH = clogb2(RANKS);
518    localparam DATA_BUF_OFFSET_WIDTH = 1;
519    localparam MC_ERR_ADDR_WIDTH = ((CS_WIDTH == 1) ? 0 : RANK_WIDTH)
520                                   + BANK_WIDTH + ROW_WIDTH + COL_WIDTH
521                                   + DATA_BUF_OFFSET_WIDTH;
522    localparam tPRDI                 = 1_000_000;
523                                       // memory tPRDI paramter in pS.
524    localparam PAYLOAD_WIDTH         = (ECC_TEST == "OFF") ? DATA_WIDTH : DQ_WIDTH;
525    localparam BURST_LENGTH          = STR_TO_INT(BURST_MODE);
526    localparam APP_DATA_WIDTH        = 2 * nCK_PER_CLK * PAYLOAD_WIDTH;
527    localparam APP_MASK_WIDTH        = APP_DATA_WIDTH / 8;
528
529    //***************************************************************************
530    // Traffic Gen related parameters (derived)
531    //***************************************************************************
532    localparam MASK_SIZE             = DATA_WIDTH/8;
```

各种wire定义

你有兴趣研究不？

我是没兴趣

```
532   localparam MASK_SIZE             = DATA_WIDTH/8;
533
534
535   // Wire declarations
536
537   wire [2*nCK_PER_CLK-1:0]              app_ecc_multiple_err;
538   wire [47:0]                           traffic_wr_data_counts;
539   wire [47:0]                           traffic_rd_data_counts;
540   wire [ADDR_WIDTH-1:0]                 app_addr;
541   wire [2:0]                            app_cmd;
542   wire                                  app_en;
543   wire                                  app_rdy;
544   wire [APP_DATA_WIDTH-1:0]             app_rd_data;
545   wire                                  app_rd_data_end;
546   wire                                  app_rd_data_valid;
547   wire [APP_DATA_WIDTH-1:0]             app_wdf_data;
548   wire                                  app_wdf_end;
549   wire [APP_MASK_WIDTH-1:0]             app_wdf_mask;
550   wire                                  app_wdf_rdy;
551   wire                                  app_sr_req;
552   wire                                  app_sr_active;
553   wire                                  app_ref_req;
554   wire                                  app_ref_ack;
555   wire                                  app_zq_req;
556   wire                                  app_zq_ack;
557   wire                                  app_wdf_wren;
558   wire [64 + (2*APP_DATA_WIDTH - 1):0]      error_status;
559   wire                                  modify_enable_sel;
560   wire [2:0]                            data_mode_manual_sel;
561   wire [2:0]                            addr_mode_manual_sel;
562   wire [APP_DATA_WIDTH-1:0]             cmp_data;
563   reg [63:0]                            cmp_data_r;
564   wire                                  cmp_data_valid;
565   wire [PAYLOAD_WIDTH/8-1:0]            dq_error_bytelane_cmp;
566
567   wire                                  clk;
568   wire                                  rst;
569
570   wire                                  vio_modify_enable;
571   wire [3:0]                            vio_data_mode_value;
572   wire                                  vio_pause_traffic;
573   wire [2:0]                            vio_addr_mode_value;
574   wire [3:0]                            vio_instr_mode_value;
575   wire [1:0]                            vio_bl_mode_value;
576   wire [7:0]                            vio_fixed_bl_value;
577   wire [2:0]                            vio_fixed_instr_value;
578   wire                                  vio_data_mask_gen;
579
580 //***************************************************************************
```

终于开始实例化DDR3了

看见DDR3 右边的#号了没？

这说明下面这些都不是管脚，
而是配置用的参数。

你继续不用改

这都六百多行了，
你还是啥也不用改。

```
595
596   DDR3 #
597     (
598
599       .TCQ                            (TCQ),
600       .ADDR_CMD_MODE                  (ADDR_CMD_MODE),
601       .AL                             (AL),
602       .PAYLOAD_WIDTH                  (PAYLOAD_WIDTH),
603       .BANK_WIDTH                     (BANK_WIDTH),
604       .BURST_MODE                     (BURST_MODE),
605       .BURST_TYPE                     (BURST_TYPE),
606       .CA_MIRROR                      (CA_MIRROR),
607       .CK_WIDTH                       (CK_WIDTH),
608       .COL_WIDTH                      (COL_WIDTH),
609       .CMD_PIPE_PLUS1                 (CMD_PIPE_PLUS1),
610       .CS_WIDTH                       (CS_WIDTH),
611       .nCS_PER_RANK                   (nCS_PER_RANK),
612       .CKE_WIDTH                      (CKE_WIDTH),
613       .DATA_WIDTH                     (DATA_WIDTH),
614       .DATA_BUF_ADDR_WIDTH            (DATA_BUF_ADDR_WIDTH),
615       .DDR2_DQSN_ENABLE               (DDR2_DQSN_ENABLE),
616       .DQ_CNT_WIDTH                   (DQ_CNT_WIDTH),
617       .DQ_PER_DM                      (DQ_PER_DM),
618       .DQ_WIDTH                       (DQ_WIDTH),
619       .DQS_CNT_WIDTH                  (DQS_CNT_WIDTH),
620       .DQS_WIDTH                      (DQS_WIDTH),
621       .DRAM_WIDTH                     (DRAM_WIDTH),
622       .ECC                            (ECC),
623       .ECC_WIDTH                      (ECC_WIDTH),
624       .ECC_TEST                       (ECC_TEST),
625       .MC_ERR_ADDR_WIDTH              (MC_ERR_ADDR_WIDTH),
626       .nAL                            (nAL),
627       .nBANK_MACHS                    (nBANK_MACHS),
628       .XC7K325T_REV_1X                (XC7K325T_REV_1X),
629       .PART_REV                       (PART_REV),
630       .CKE_ODT_AUX                    (CKE_ODT_AUX),
631       .ORDERING                       (ORDERING),
632       .OUTPUT_DRV                     (OUTPUT_DRV),
633       .IBUF_LPWR_MODE                 (IBUF_LPWR_MODE),
634       .IODELAY_HP_MODE                (IODELAY_HP_MODE),
635       .DATA_IO_IDLE_PWRDWN            (DATA_IO_IDLE_PWRDWN),
636       .DATA_IO_PRIM_TYPE              (DATA_IO_PRIM_TYPE),
637       .REG_CTRL                       (REG_CTRL),
638       .RTT_NOM                        (RTT_NOM),
639       .RTT_WR                         (RTT_WR),
640       .CL                             (CL),
641       .CWL                            (CWL),
642       .tCKE                           (tCKE),
643       .tFAW                           (tFAW),
```

唉呀妈呀，DDR3实例化的实体总算找到了，就叫做
u_DDR3
找到没，我这里是747行

接下来你要改动的，其实只有区区几行

那就是

769行Application interface开始的几个ports
从770行的app_addr开始
到775行的app_wdf_wren结束

一共六行

此外，因为你之前选了data mask，所以790行有个app_wdf_mask
这一行的赋值你可以直接改成零。
这个值来自traffic gen
traffic gen你是要删掉的，删掉之后没赋值的app_wdf_mask自然被默认成零。

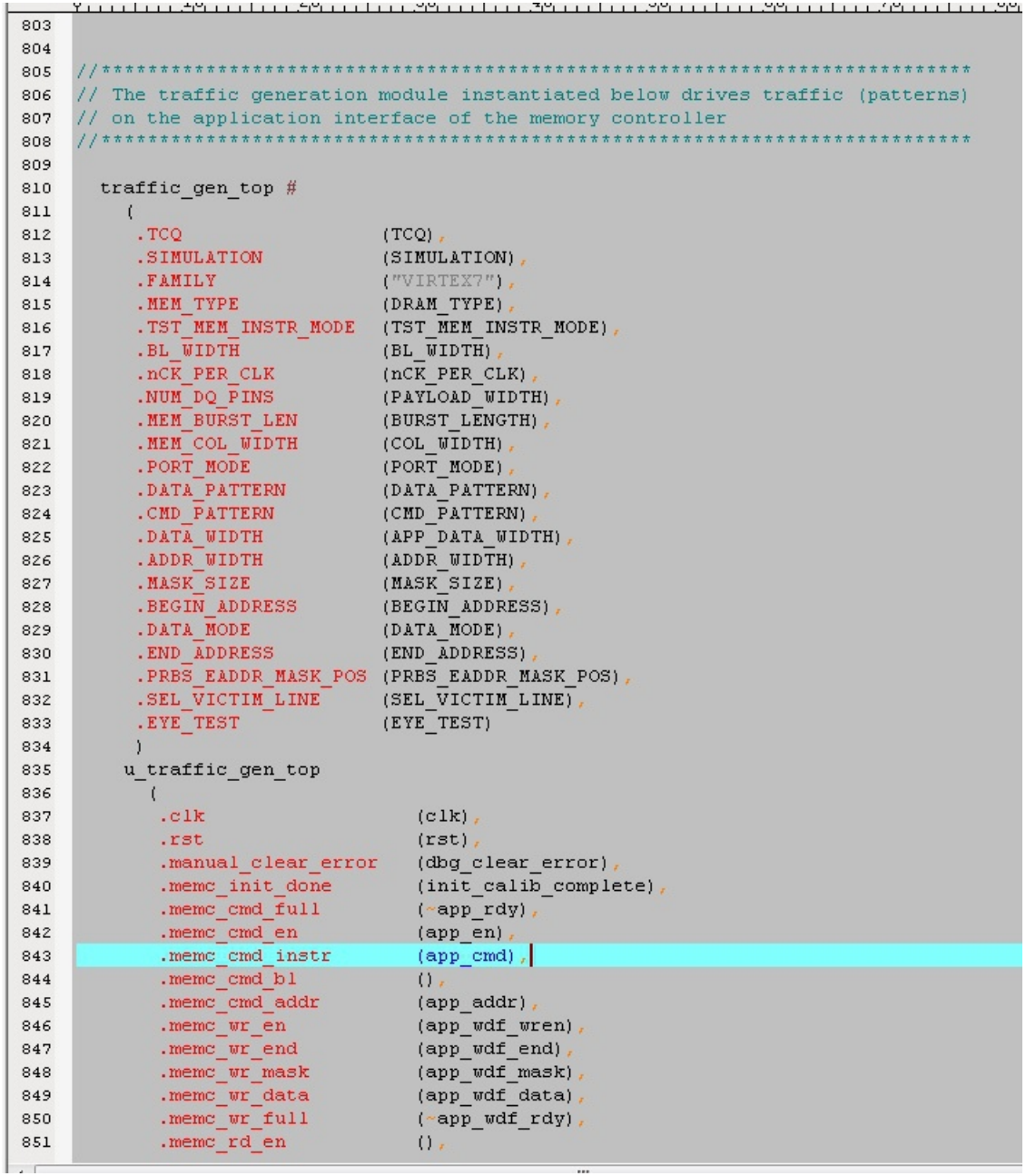
814

```
745       .RST_ACT_LOW                    (RST_ACT_LOW)
746       )
747      u_DDR3
748        (
749
750
751   // Memory interface ports
752         .ddr3_addr                    (ddr3_addr),
753         .ddr3_ba                      (ddr3_ba),
754         .ddr3_cas_n                   (ddr3_cas_n),
755         .ddr3_ck_n                    (ddr3_ck_n),
756         .ddr3_ck_p                    (ddr3_ck_p),
757         .ddr3_cke                     (ddr3_cke),
758         .ddr3_ras_n                   (ddr3_ras_n),
759         .ddr3_reset_n                 (ddr3_reset_n),
760         .ddr3_we_n                    (ddr3_we_n),
761         .ddr3_dq                      (ddr3_dq),
762         .ddr3_dqs_n                   (ddr3_dqs_n),
763         .ddr3_dqs_p                   (ddr3_dqs_p),
764         .init_calib_complete          (init_calib_complete),
765
766         .ddr3_cs_n                    (ddr3_cs_n),
767         .ddr3_dm                      (ddr3_dm),
768         .ddr3_odt                     (ddr3_odt),
769   // Application interface ports
770         .app_addr                     (app_addr),
771         .app_cmd                      (app_cmd),
772         .app_en                       (app_en),
773         .app_wdf_data                 (app_wdf_data),
774         .app_wdf_end                  (app_wdf_end),
775         .app_wdf_wren                 (app_wdf_wren),
776         .app_rd_data                  (app_rd_data),
777         .app_rd_data_end              (app_rd_data_end),
778         .app_rd_data_valid            (app_rd_data_valid),
779         .app_rdy                      (app_rdy),
780         .app_wdf_rdy                  (app_wdf_rdy),
781         .app_sr_req                   (1'b0),
782         .app_sr_active                (app_sr_active),
783         .app_ref_req                  (1'b0),
784         .app_ref_ack                  (app_ref_ack),
785         .app_zq_req                   (1'b0),
786         .app_zq_ack                   (app_zq_ack),
787         .ui_clk                       (clk),
788         .ui_clk_sync_rst              (rst),
789
790         .app_wdf_mask                 (app_wdf_mask),
791
792
793   // System Clock Ports
```

DDR实例化完了之后就是traffic gen的实例化

这个traffic gen对设计来讲完全没用，连参考价值都没有

删了删了都删了

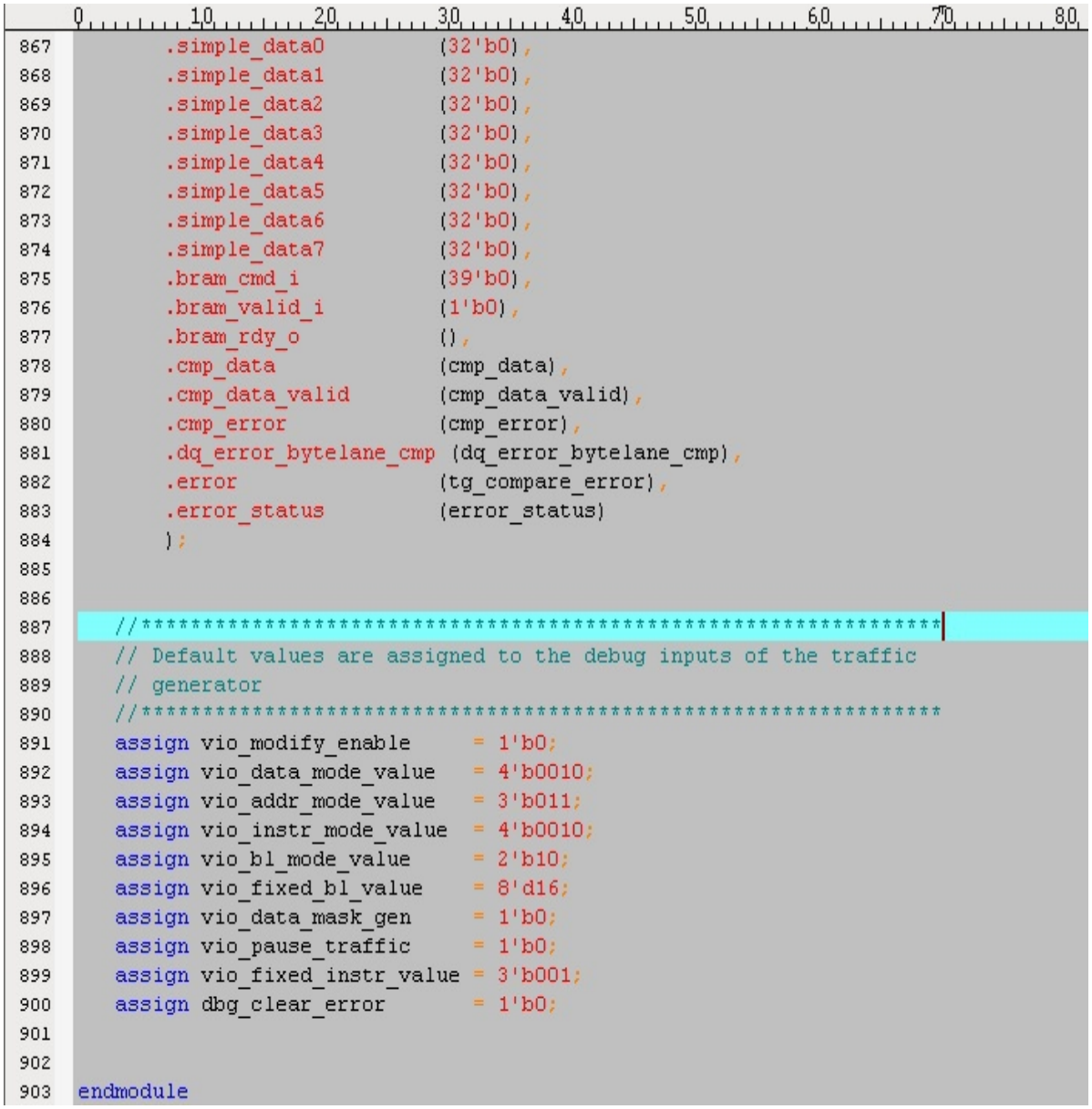


```
803
804
805 //***************************************************************************
806 // The traffic generation module instantiated below drives traffic (patterns)
807 // on the application interface of the memory controller
808 //***************************************************************************
809
810   traffic_gen_top #
811     (
812      .TCQ                  (TCQ),
813      .SIMULATION           (SIMULATION),
814      .FAMILY               ("VIRTEX7"),
815      .MEM_TYPE             (DRAM_TYPE),
816      .TST_MEM_INSTR_MODE   (TST_MEM_INSTR_MODE),
817      .BL_WIDTH             (BL_WIDTH),
818      .nCK_PER_CLK          (nCK_PER_CLK),
819      .NUM_DQ_PINS          (PAYLOAD_WIDTH),
820      .MEM_BURST_LEN        (BURST_LENGTH),
821      .MEM_COL_WIDTH        (COL_WIDTH),
822      .PORT_MODE            (PORT_MODE),
823      .DATA_PATTERN         (DATA_PATTERN),
824      .CMD_PATTERN          (CMD_PATTERN),
825      .DATA_WIDTH           (APP_DATA_WIDTH),
826      .ADDR_WIDTH           (ADDR_WIDTH),
827      .MASK_SIZE            (MASK_SIZE),
828      .BEGIN_ADDRESS        (BEGIN_ADDRESS),
829      .DATA_MODE            (DATA_MODE),
830      .END_ADDRESS          (END_ADDRESS),
831      .PRBS_EADDR_MASK_POS  (PRBS_EADDR_MASK_POS),
832      .SEL_VICTIM_LINE      (SEL_VICTIM_LINE),
833      .EYE_TEST             (EYE_TEST)
834      )
835     u_traffic_gen_top
836       (
837        .clk                  (clk),
838        .rst                  (rst),
839        .manual_clear_error   (dbg_clear_error),
840        .memc_init_done       (init_calib_complete),
841        .memc_cmd_full        (~app_rdy),
842        .memc_cmd_en          (app_en),
843        .memc_cmd_instr       (app_cmd),
844        .memc_cmd_bl          (),
845        .memc_cmd_addr        (app_addr),
846        .memc_wr_en           (app_wdf_wren),
847        .memc_wr_end          (app_wdf_end),
848        .memc_wr_mask         (app_wdf_mask),
849        .memc_wr_data         (app_wdf_data),
850        .memc_wr_full         (~app_wdf_rdy),
851        .memc_rd_en           (),
```

891行这一对都是traffic gen相关参数

没用的，都删了

记得别把903行endmodule一起删了啊，嘿嘿



```
867         .simple_data0           (32'b0),
868         .simple_data1           (32'b0),
869         .simple_data2           (32'b0),
870         .simple_data3           (32'b0),
871         .simple_data4           (32'b0),
872         .simple_data5           (32'b0),
873         .simple_data6           (32'b0),
874         .simple_data7           (32'b0),
875         .bram_cmd_i             (39'b0),
876         .bram_valid_i           (1'b0),
877         .bram_rdy_o             (),
878         .cmp_data               (cmp_data),
879         .cmp_data_valid         (cmp_data_valid),
880         .cmp_error              (cmp_error),
881         .dq_error_bytelane_cmp (dq_error_bytelane_cmp),
882         .error                  (tg_compare_error),
883         .error_status           (error_status)
884         );
885
886
887    //***************************************************************
888    // Default values are assigned to the debug inputs of the traffic
889    // generator
890    //***************************************************************
891    assign vio_modify_enable     = 1'b0;
892    assign vio_data_mode_value   = 4'b0010;
893    assign vio_addr_mode_value   = 3'b011;
894    assign vio_instr_mode_value  = 4'b0010;
895    assign vio_bl_mode_value     = 2'b10;
896    assign vio_fixed_bl_value    = 8'd16;
897    assign vio_data_mask_gen     = 1'b0;
898    assign vio_pause_traffic     = 1'b0;
899    assign vio_fixed_instr_value = 3'b001;
900    assign dbg_clear_error       = 1'b0;
901
902
903 endmodule
```

关于example_top.v

这是综合工程的顶层

但在testbench里面它不是顶层

因为testbench里面和它同级的还有DDR的仿真模型

testbench的顶层是
E:\DDR3\DDR3\ipcore_dir\DDR3\example_design\sim\sim_tb_top.v

这个sim_tb_top.v呢，你打开研究一下就知道怎么回事儿
结论就是，你不需要去改动他

整个设计过程里要你改动的，就只有example_top.v，而且其实只有
example_top.v的Application interface的几个ports
从770行的app_addr开始
到775行的app_wdf_wren结束

如果你删了traffic gen，那么app_wdf_mask都可以不改，因为默认反正是零么。

example_top.v的Application interface的几个ports
从770行的app_addr开始，到775行的app_wdf_wren结束
这都是些什么呢
我们来研究一下

app_addr
操作地址，按照结构从高位到低位是
rank + bank + row + column

app_cmd
操作命令，其实你只需要用到3'b000（写入）和3'b001（读出）
注意，要和操作地址同时出现才有效。

app_en
操作地址app_addr的使能，只有它拉高的时候，对应的app_addr才是有效的

app_wdf_data
写入的数据接口

app_wdf_end
理论上应该有点用，但是实际你只要让它跟app_wdf_wren一样就行了

app_wdf_wren
写入的数据接口app_wdf_data的使能，
只有它拉高的时候，对应的app_wdf_data才是有效的

为了方便理解，对这几个信号我们重新排序一下吧

app_cmd　　（你总要先确认你想要写还是想要读吧）
操作命令，其实你只需要用到3'b000（写入）和3'b001（读出）
要和操作地址同时出现才有效。

app_addr　　（往哪儿写，从哪儿读？）
操作地址，按照结构从高位到低位是
rank + bank + row + column

app_en　　（确认地址线上的地址有效，不能初始值都一直有效吧）
操作地址app_addr的使能，只有它拉高的时候，对应的app_addr才是有效的

app_wdf_data　　（要写的话，你得有料不是）
写入的数据接口

app_wdf_wren　　（那也不能什么料都往里倒不是）
写入的数据接口app_wdf_data的使能，
只有它拉高的时候，对应的app_wdf_data才是有效的

app_wdf_end　　（要你作甚，一句app_wdf_end = app_wdf_wren 搞定）
理论上应该有点用，但是实际你只要让它跟app_wdf_wren一样就行了

所以你新建了一个.v文件
比如我们叫它ddr3_app_control.v
它将输出以下六个信号：

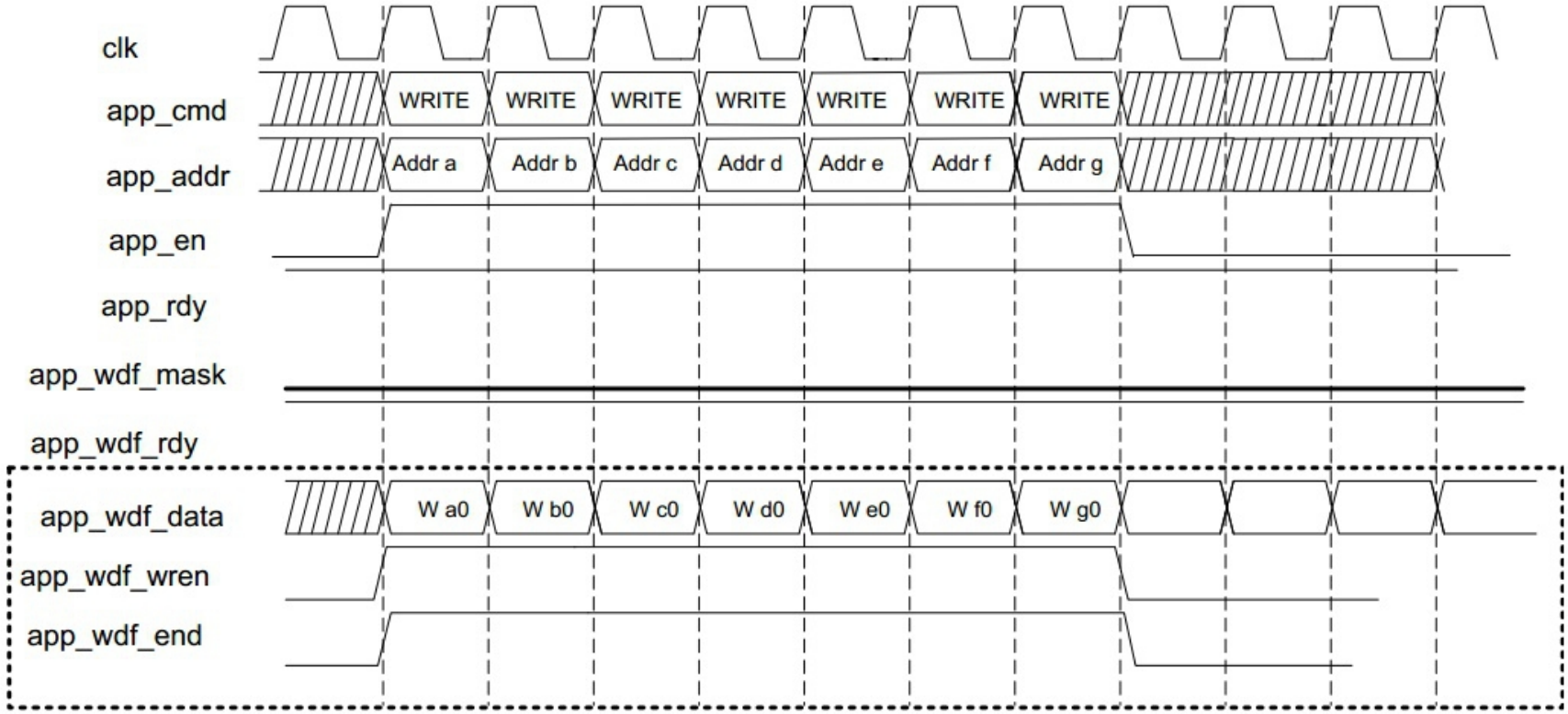
app_cmd
app_addr
app_en
app_wdf_data
app_wdf_wren
app_wdf_end

这六个信号就全靠你写代码来产生了
确切的说是五个信号

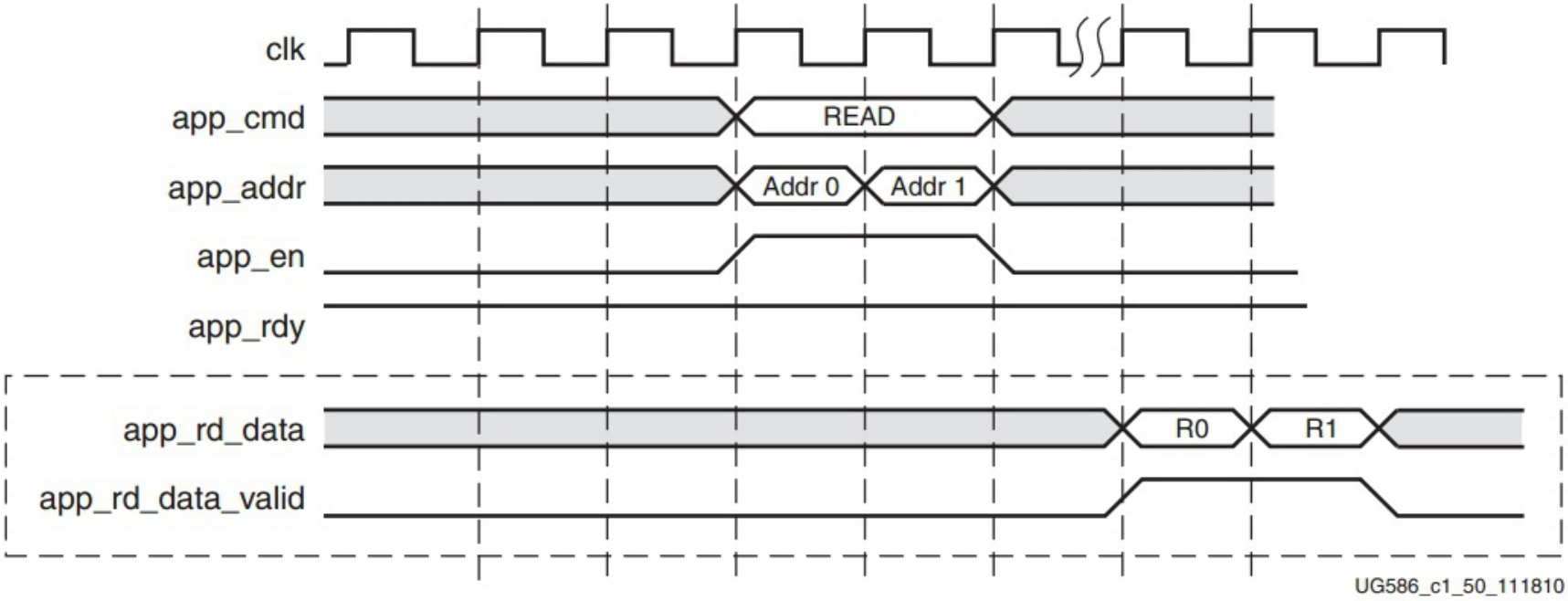
因为app_wdf_end = app_wdf_wren啊，嘿嘿

这五个信号，应该满足怎么的时序关系，才能操作DDR呢？

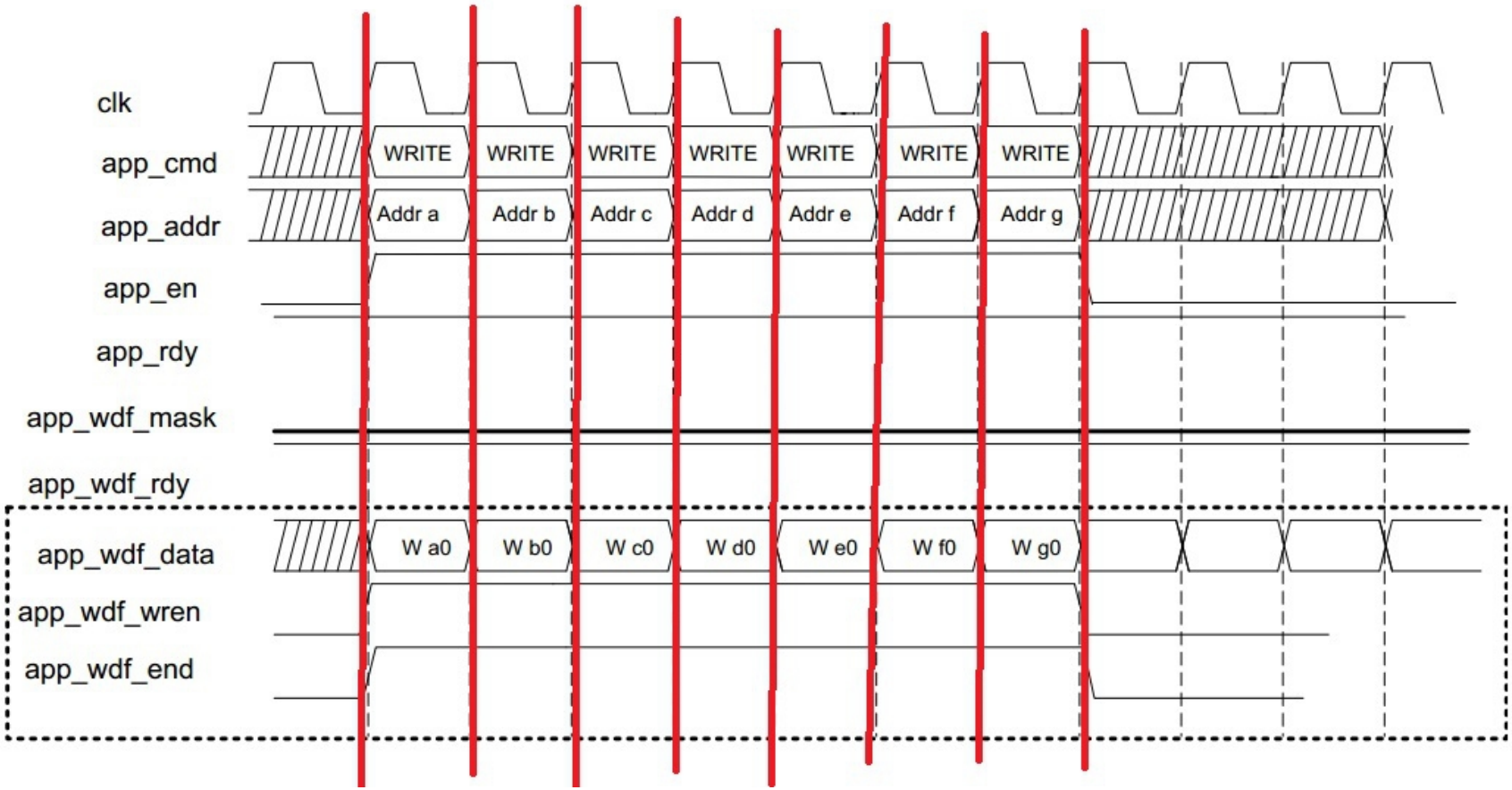
想写你就这样写


clk
app_cmd
app_addr
app_en
app_rdy
app_wdf_mask
app_wdf_rdy
app_wdf_data
app_wdf_wren
app_wdf_end
WRITE
WRITE
WRITE
WRITE
WRITE
WRITE
WRITE
Addr a
Addr b
Addr c
Addr d
Addr e
Addr f
Addr g
W a0
W b0
W c0
W d0
W e0
W f0
W g0

想读你就这样读


clk
app_cmd
READ
app_addr
Addr 0
Addr 1
app_en
app_rdy
app_rd_data
R0
R1
app_rd_data_valid
UG586_c1_50_111810

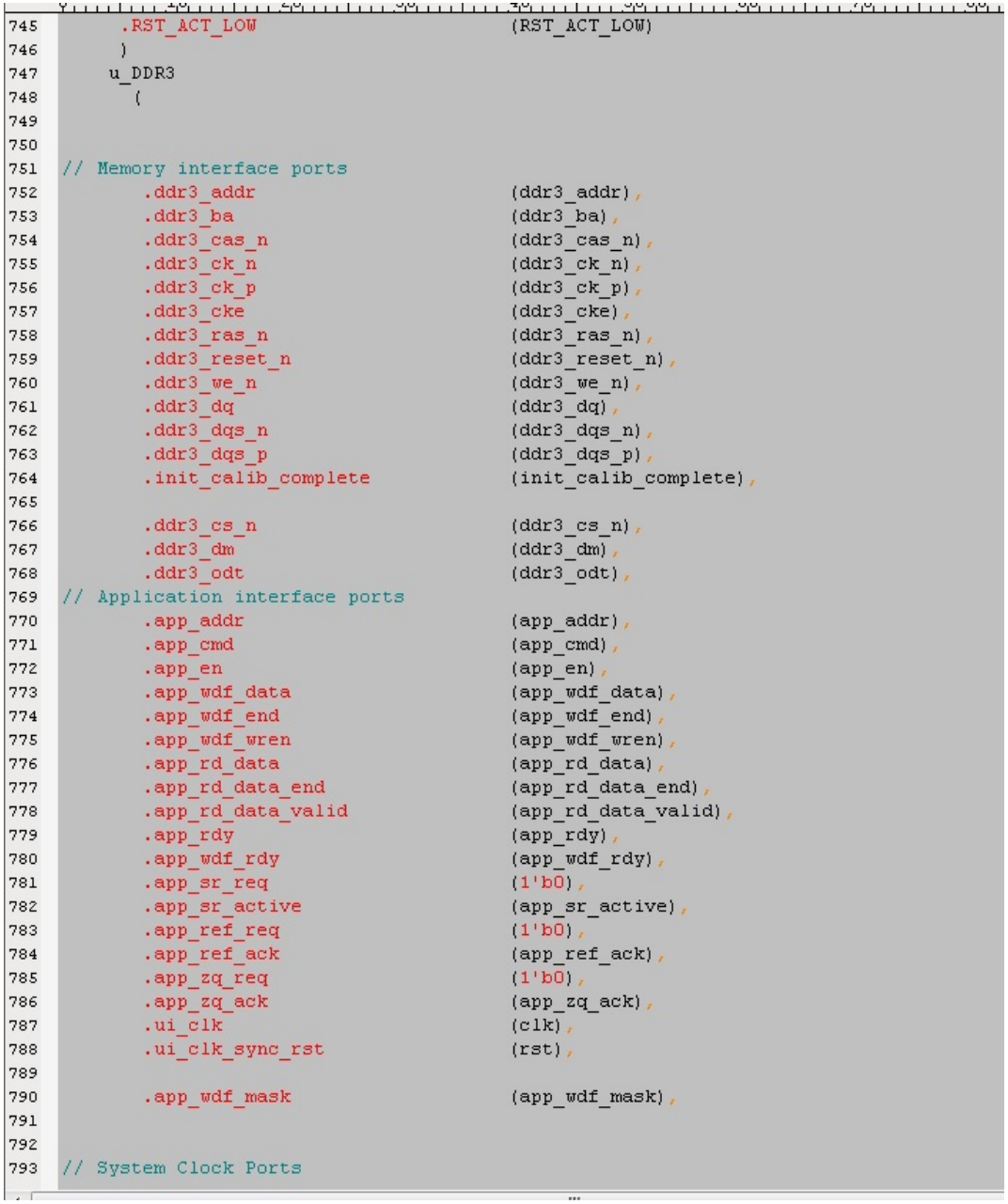
什么？刚才没看明白？还是难以想象这接口这么简单？
要不要我再说明白一点啊？画几条辅助线？
简单到就一条，你全部对齐就可以了。
当然，前提是在app_rdy和app_wdf_rdy都拉高的情况下
又什么？你对app_rdy和app_wdf_rdy都没啥印象？拜托，你记性不好吧？
要不要买点脑白金补补啊？



之前已经出现过的一张图

请问779行和780行分别是啥？

要不要往前翻一翻啊？



```
745       .RST_ACT_LOW                    (RST_ACT_LOW)
746       )
747      u_DDR3
748        (
749
750
751   // Memory interface ports
752         .ddr3_addr                    (ddr3_addr),
753         .ddr3_ba                      (ddr3_ba),
754         .ddr3_cas_n                   (ddr3_cas_n),
755         .ddr3_ck_n                    (ddr3_ck_n),
756         .ddr3_ck_p                    (ddr3_ck_p),
757         .ddr3_cke                     (ddr3_cke),
758         .ddr3_ras_n                   (ddr3_ras_n),
759         .ddr3_reset_n                 (ddr3_reset_n),
760         .ddr3_we_n                    (ddr3_we_n),
761         .ddr3_dq                      (ddr3_dq),
762         .ddr3_dqs_n                   (ddr3_dqs_n),
763         .ddr3_dqs_p                   (ddr3_dqs_p),
764         .init_calib_complete          (init_calib_complete),
765
766         .ddr3_cs_n                    (ddr3_cs_n),
767         .ddr3_dm                      (ddr3_dm),
768         .ddr3_odt                     (ddr3_odt),
769   // Application interface ports
770         .app_addr                     (app_addr),
771         .app_cmd                      (app_cmd),
772         .app_en                       (app_en),
773         .app_wdf_data                 (app_wdf_data),
774         .app_wdf_end                  (app_wdf_end),
775         .app_wdf_wren                 (app_wdf_wren),
776         .app_rd_data                  (app_rd_data),
777         .app_rd_data_end              (app_rd_data_end),
778         .app_rd_data_valid            (app_rd_data_valid),
779         .app_rdy                      (app_rdy),
780         .app_wdf_rdy                  (app_wdf_rdy),
781         .app_sr_req                   (1'b0),
782         .app_sr_active                (app_sr_active),
783         .app_ref_req                  (1'b0),
784         .app_ref_ack                  (app_ref_ack),
785         .app_zq_req                   (1'b0),
786         .app_zq_ack                   (app_zq_ack),
787         .ui_clk                       (clk),
788         .ui_clk_sync_rst              (rst),
789
790         .app_wdf_mask                 (app_wdf_mask),
791
792
793   // System Clock Ports
```

所以，现在似乎写入的动作变得复杂了一点，你得多考虑两个信号了。
是不是觉得头有点晕啊？

来，重新理一理，要写DDR，需要考虑哪些信号？

app_cmd　　（你总要先确认你想要写还是想要读吧）
操作命令，其实你只需要用到3'b000（写入）和3'b001（读出）

app_addr　　（往哪儿写，从哪儿读？）
操作地址，按照结构从高位到低位是
rank + bank + row + column

app_en　　（确认地址线上的地址有效，不能初始值都一直有效吧）
操作地址app_addr的使能，只有它拉高的时候，对应的app_addr才是有效的

app_wdf_data　　（要写的话，你得有料不是）
写入的数据接口

app_wdf_wren　　（那也不能什么料都往里倒不是）
写入的数据接口app_wdf_data的使能，
只有它拉高的时候，对应的app_wdf_data才是有效的

app_wdf_end　　（要你作甚，一句app_wdf_end = app_wdf_wren 搞定）
理论上应该有点用，但是实际你只要让它跟app_wdf_wren一样就行了

==========华丽丽的分割线，别忘了关注下面两个信号==========

app_rdy
（DDR告诉你，你在app_rdy拉高的时候拉高app_en，地址app_addr 才是有效的）

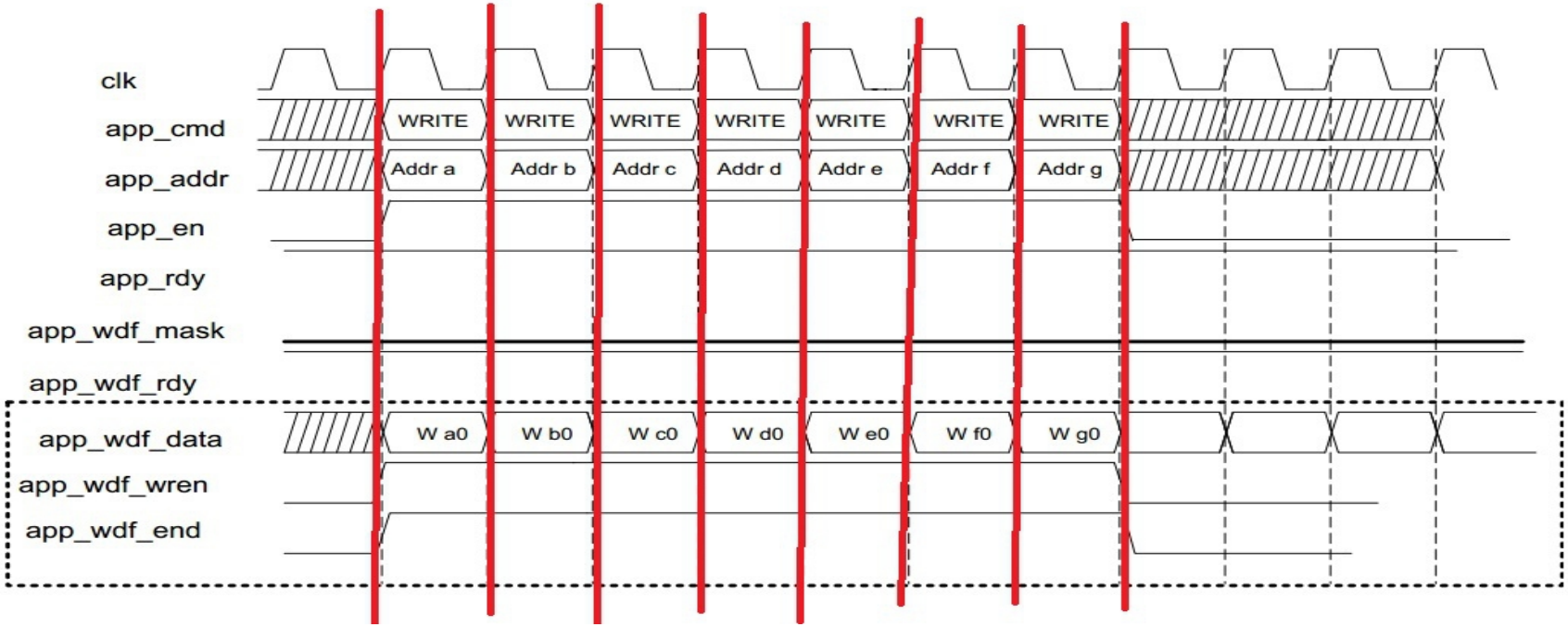
app_wdf_rdy
（DDR又告诉你，你在app_wdf_rdy拉高的时候拉高app_wdf_wren，写入数据app_wdf_data 才是有效的）

所以现在你知道了，写入操作是两套系统
一套是地址，另一套是数据

地址的内容是app_addr，它在
app_rdy（DDR控制）和app_en（你自己控制）同时拉高的时候才是有效的

数据的内容是app_wdf_data，它在
app_wdf_rdy（DDR控制）和app_wdf_wren（你自己控制）同时拉高的时候才是有效的

这两套系统在时序上完全对齐，就可以把数据成功的写入DDR了



其实吧，也不一定要完全对齐了

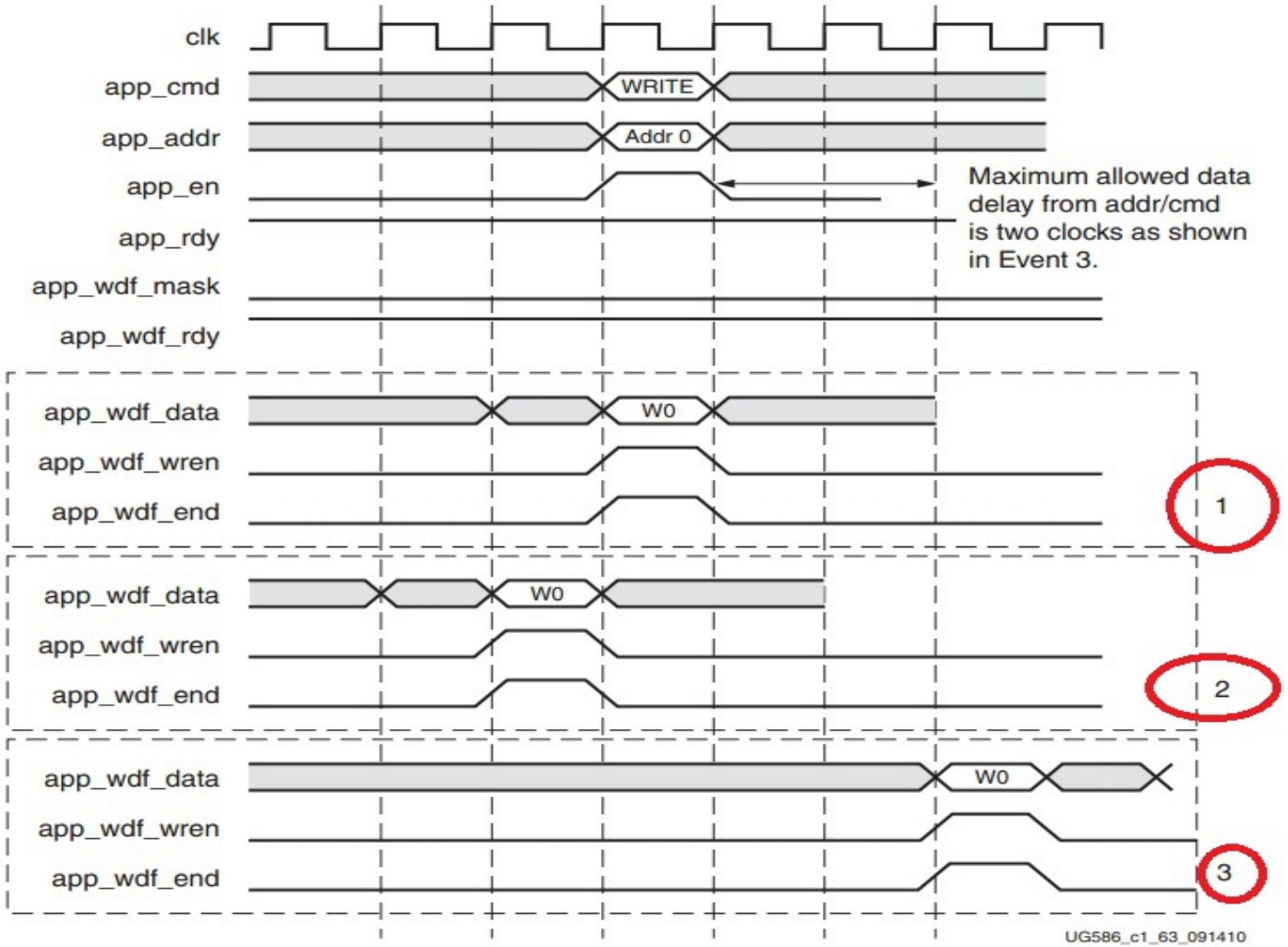
如果地址系统和数据系统（注意，是数据和地址两套系统之间，单位是系统）稍有不对齐，这IP也是可以凑合的
如下图

这图上显示了123三种情况。

1是严格对齐
2和3告诉你系统和系统之间稍微有点前后偏差，也是可以接受的。

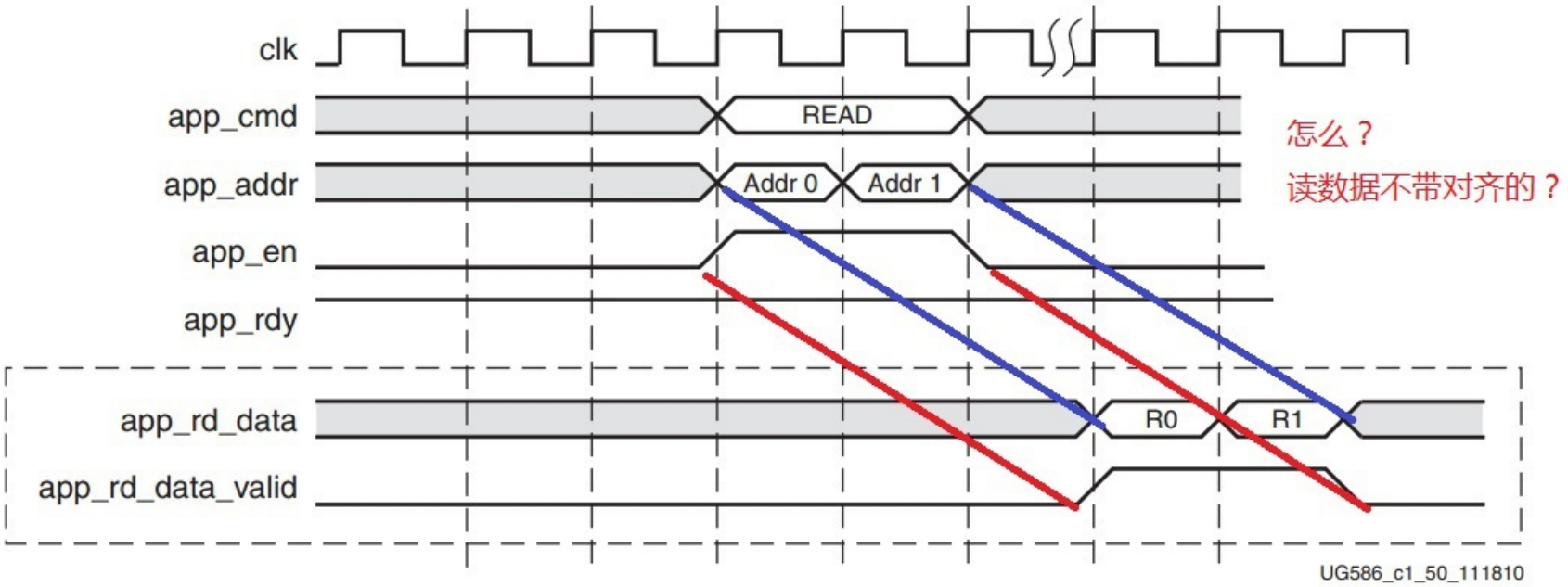
具体去看手册吧~

手册下载地址下一页再给你一次。



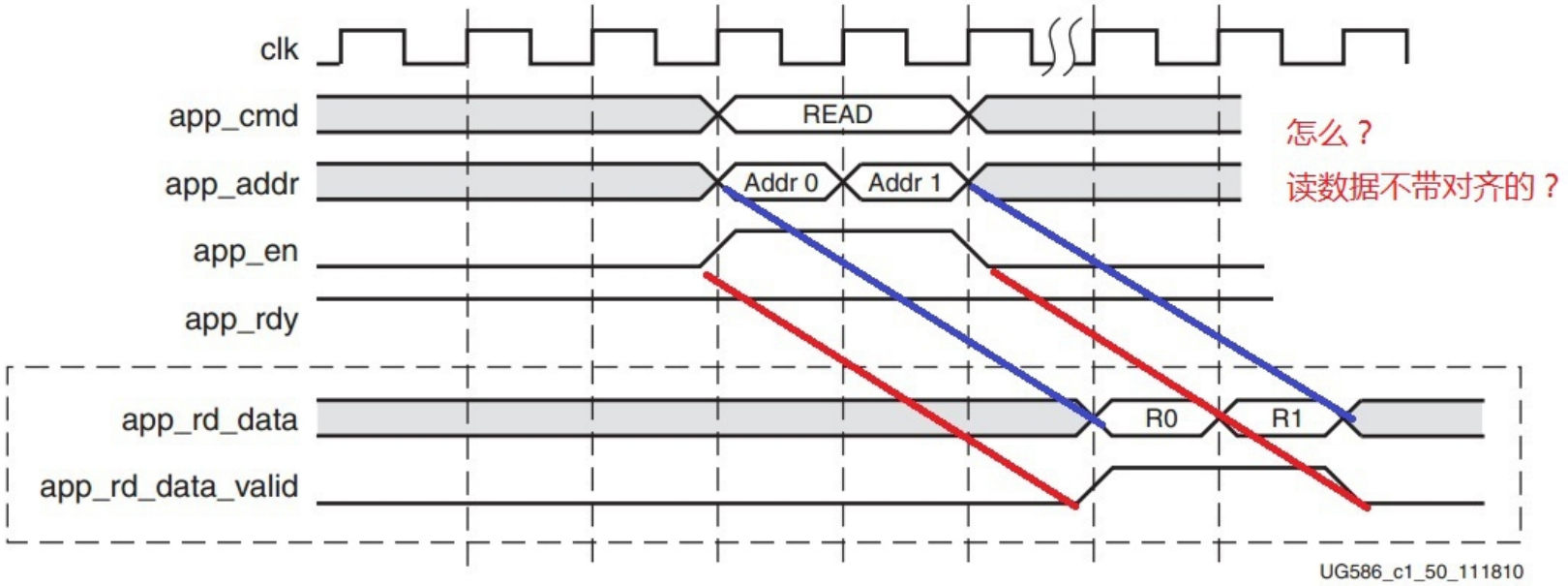
就这里下载：
http://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v1_5/ug586_7Series_MIS.pdf

包括下面这个读数据的图也是来自这个手册



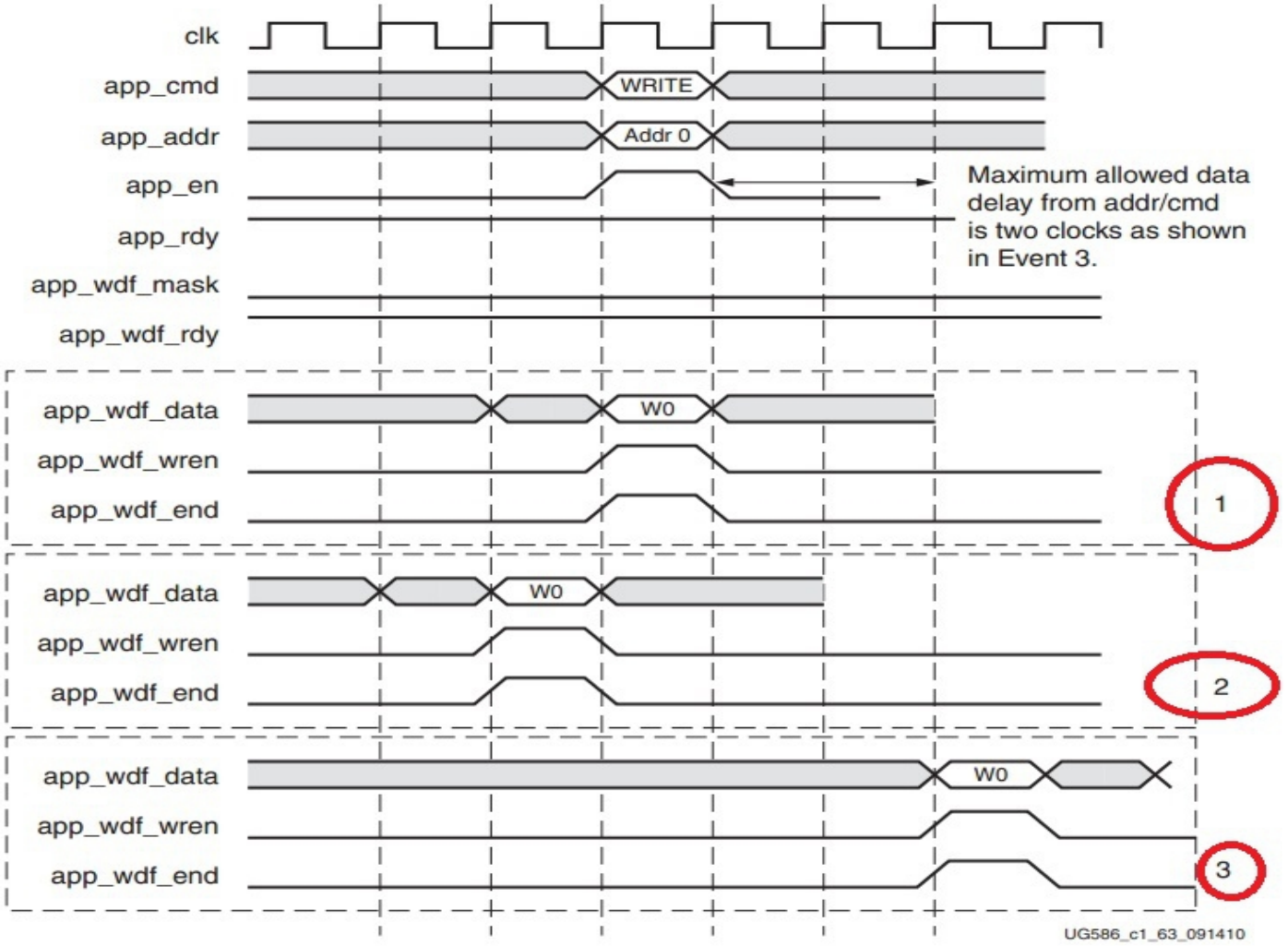
那么我反问你，读数据为什么要对齐？
你给地址，DDR隔一阵再给你数据，你哪里疼哪里痒？
人家DDR也需要反应时间的好不好？

DDR给你的信号是两个，app_rd_data和app_rd_data_valid
顾名思义，数据和数据使能。
而你给DDR的数据也是两个，地址和读命令。
当然同样在app_rdy和app_en拉高的时候，地址才有效。
这回不需要考虑app_wdf_rdy了——真的，因为你这回是读，又不是写。



怎么？
读数据不带对齐的？

最后再说两个注意点
第一个注意点，
命令和地址其实是一样的处理，是对齐绑定在一起的，都属于地址系统。
看下面这个图就很明白了，刚出现过的。
命令和地址系统对齐绑定，而不和数据系统绑定。



第二个注意点，
地址的算法

我举一个例子
比如4G的笔记本内存条，双RANK
（几个rank就是几面贴颗粒，双rank就是两面都有贴DDR颗粒）
每一面贴8个颗粒，那就是每一面8个bank（bank就是颗粒，DDR芯片）
这种地址怎么算？
经本人实际验证，4G是这么出来的：
column 宽度是10个bit
row宽度是 15个bit
bank 宽度是3个bit
rank宽度是1个bit
这样app_addr的位宽就是把以上全部加起来
10 + 15 + 3 + 1 = 29 个 bit

对每一个bank（颗粒）来说，一共有2^（10 + 15）个地址
这里^是verilog取指数的符号，就是2的25次方，写VHDL的将就看吧
2的25次方是多少？
结果是33554432（十进制）个地址
每个地址可以存64个bit的数据，那么一共就是
33554432 x 64 = 2147483648个bit
（翻下页）

每个地址可以存64个bit的数据，那么每个颗粒（bank）里面一共就是
33554432 x 64 = 2147483648个bit
不过呢，我们说的4G内存条，那是4G byte，不是4Gbit
我们换算成byte
2147483648个bit = 268435456个byte
约等于 268MB

之前说了，两个rank（内存条的两面），每个rank贴8个bank（颗粒）
一共就是16个颗粒
那么内存条的总容量就是 268MB x 16 = 4.3GB

这就是4G内存条的由来，并非很多人原先以为的4.096GB

一般来讲，如果内存条上贴满16个颗粒，那么4G和2G，8G的区别就是
row地址的宽度
这个例子里，row的宽度是15个bit
那么16颗粒的2G内存条，row宽度就是14个bit
那么16颗粒的8G内存条，row宽度就是16个bit
8G的DDR3内存条只有一种，那就是16颗粒的
4G的DDR3内存条也有8颗粒的，每个rank有4个bank而不是8个

用的时候注意区别对待。

关于DDR用户接口，了解这些就可以开始动手写代码了。