SDRAM 驱动篇之简易 SDRAM 控制器的 verilog 代码实现

FPGA 学习笔记 Kevin 2 年前 (2016-01-17) 8899℃ 0 评论

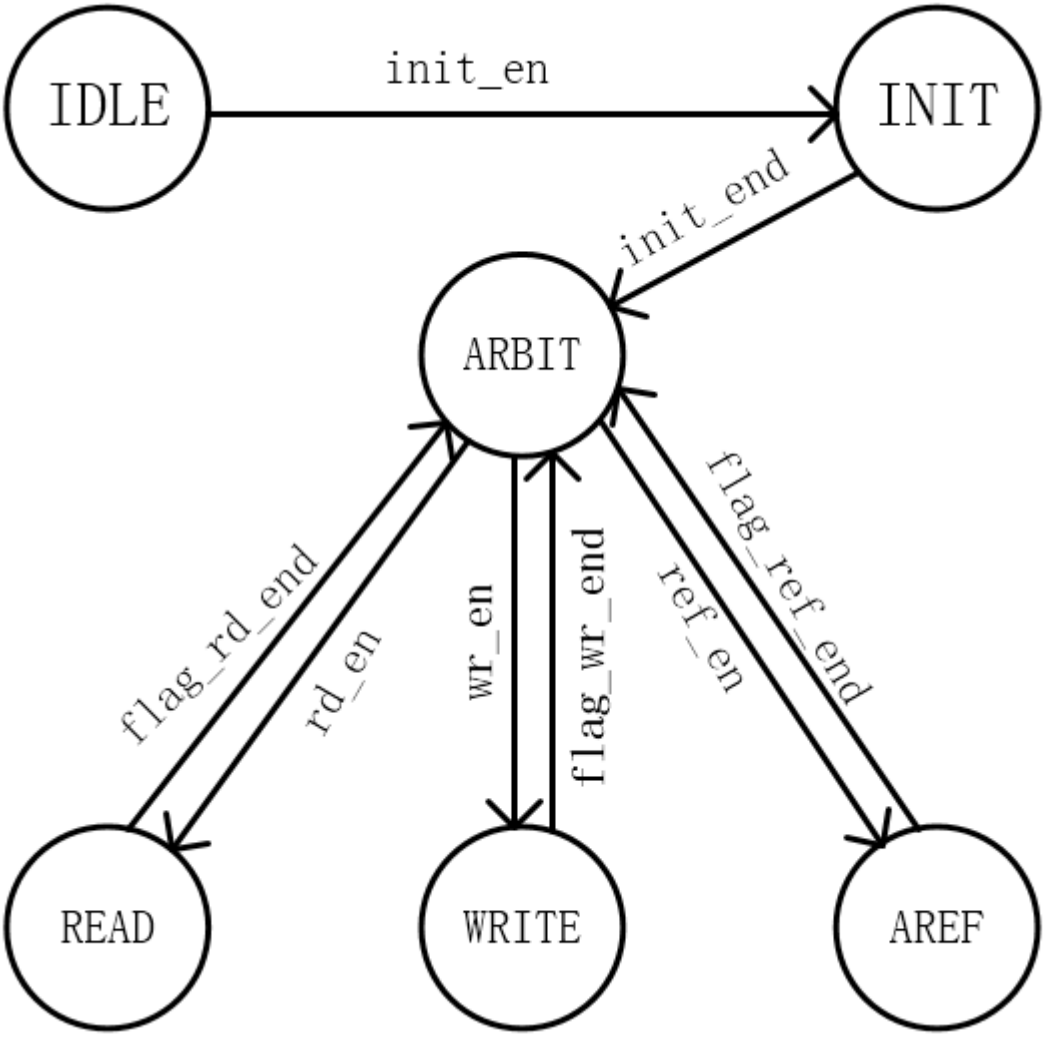
在 Kevin 写的上一篇博文《SDRAM 理论篇之基础知识及操作时序》中，已经把 SDRAM 工作的基本原理和 SDRAM 初始化、读、写及自动刷新操作的时序讲清楚了，在这一片博文中，Kevin 来根据在上一篇博文中分析的思路来把写一个简单的 SDRAM 控制器。

我们在上一篇博文中提到了这样一个问题，SDRAM 是每隔 15us 进行刷新一次，但是如果当 SDRAM 需要进行刷新时，而 SDRAM 正在写数据，这两个操作之间怎么进行协调呢？因为我们是肯定需要保证写的数据不能丢失，所以，我们可以考虑这样来做：如果刷新的时间到了，先让写操作把正在写的 4 个数据（突发长度为 4）写完，然后再去进行刷新操作。而如果在执行读操作也遇到需要刷新的情况，我们也可以这样来做，先让数据读完，再去执行刷新操作。

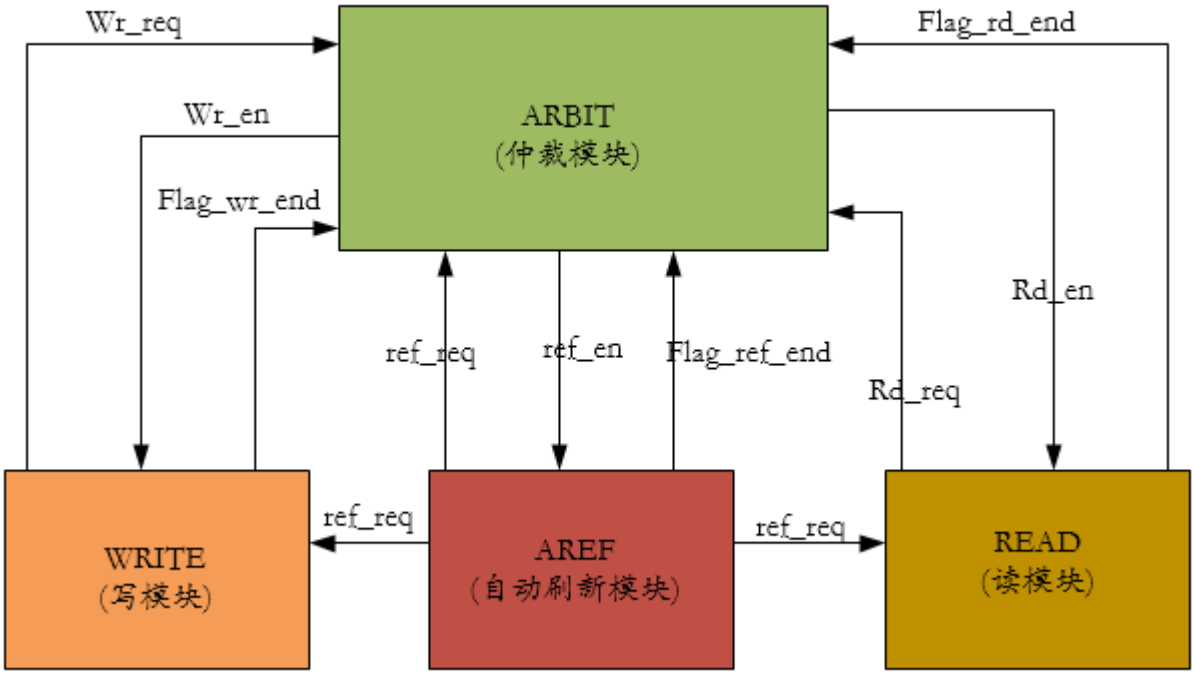
大家看完可能会想，说是这么说，那代码怎么来写呢？似乎还是没什么思路。大家可以想象一下，我们写的 SDRAM 控制器是肯定包括初始化、读操作、写操作及自动刷新这些操作的，既然这样，我们就可以给每一个操作写上一个模块独立开来，这样也便于我们每个模块的调试，显然这种思路是正确的。那怎么让我们的各个模块工作起来呢，虽然都是独立的模块，但很显然这几个模块之间又是相互关联的。就拿上面刚才说的那个情况来讲，如果 SDRAM 需要刷新了，而 SDRAM 却正在执行写操作，那我们刷新模块与写模块之间怎么进行控制呢？这个问题解决了，读模块与刷新模块之间的这个问题也可以很轻松的解决。大家不妨可以自己先想一下。

主状态机与各模块间的连线

为了解决各个模块之间不方便控制的情况，我们引入一个新的机制 ——“仲裁”机制。“仲裁”用来干什么呢？在这里边，“仲裁”相当于我们这个 SDRAM 控制器的老大，对 SDRAM 的各个操作统一协调：读、写及自动刷新都由“仲裁”来控制。说到这里，显然我们可以再写一个“仲裁”模块，既然在仲裁模块中要控制这么多操作，那自然而然的肯定想到了利用状态机。那我们的状态机怎么来设计呢？请看下图：



只给一个状态机的图，Kevin 还是觉得不够说明问题，再上一个模块之间的示意图：



在讲之前，Kevin 要给大家打一下预防针：在接下来讲的过程中，大家一定要搞清楚 Kevin 说的是模块之间连线的关系还是状态机之间跳转的关系哦。

在仲裁模块中，初始化操作完成之后便进入到了“ARBIT”仲裁状态，只有处于仲裁状态的时候，“仲裁老大”才能进行下命令。我们先来模拟一下，当状态机处于“WRITE”写状态时，如果 SDRAM 刷新的时间到了，刷新模块同时向写模块和仲裁模块发送刷新请求 ref_req 信号，当写模块接受到 ref_req 之后，写模块在写完当前 4 个数据（突发长度为 4）之后，写模块的写结束标志 flag_wr_end 拉高，然后状态机进入“ARBIT”仲裁状态，处于仲裁状态之后，此时有刷新请求 ref_req，然后状态机跳转到“AREF”状态并且仲裁模块发送 ref_en 刷新使能，然后呢，刷新模块将刷新请求信号 ref_req 拉低并给 sdram 发送刷新的命令。等刷新完毕之后，刷新模块给仲裁模块发送 flag_ref_end 刷新结束标志，状态机跳转到“ARBIT”仲裁状态。

注意了，当刷新完跳转到“ARBIT”仲裁状态之后，如果之前我们的全部数据仍然没有写完（Kevin 指的是全部数据，并不是一个突发长度的 4 个数据哦），那么此时我们仍然要给仲裁模块写请求“wr_req”，然后仲裁模块经过一系列判断之后，如果符合写操作的时机，那就给写模块一个写使能信号“wr_en”，然后跳转到“WRITE”写状态并且写模块开始工作。

对于读模块与刷新操作之间的协调，相信大家应该也能想象得到了，Kevin 在这里就不再啰嗦了。

仲裁模块（顶层模块）代码介绍

下面先看一下在我们的仲裁模块（顶层模块）中状态机的定义：

```
1.  //state
2.            always    @(posedge sclk or negedge s_rst_n)
```

```
3.                      if(s_rst_n == 1'b0)
4.                              state     <=        IDLE;
5.                      else case(state)
6.                              IDLE:
7.                                      if(key[0] == 1'b1)
8.                                              state     <=        INIT;
9.                                      else
10.                                             state     <=        IDLE;
11.                             INIT:
12.                                     if(flag_init_end == 1'b1)     //初始化结束标志
13.                                             state     <=        ARBIT;
14.                                     else
15.                                             state     <=        INIT;
16.                             ARBIT:
17.                                     if(ref_req == 1'b1) //刷新请求到来且已经写完
18.                                             state     <=        AREF;
19.                                     else if(ref_req == 1'b0 && rd_en == 1'b1)
            //默认读操作优先于写操作
20.                                             state     <=        READ;
21.                                     else if(ref_req == 1'b0 && wr_en == 1'b1)
            //无刷新请求且写请求到来
22.                                             state     <=        WRITE;
23.                                     else
24.                                             state     <=        ARBIT;
25.                             AREF:
26.                                     if(flag_ref_end == 1'b1)
27.                                             state     <=        ARBIT;
28.                                     else
29.                                             state     <=        AREF;
30.                             WRITE:
31.                                     if(flag_wr_end == 1'b1)
32.                                             state     <=        ARBIT;
33.                                     else
34.                                             state     <=        WRITE;
35.                             READ:
36.                                     if(flag_rd_end == 1'b1)
37.                                             state     <=        ARBIT;
38.                                     else
39.                                             state     <=        READ;
40.                             default:
41.                                     state     <=        IDLE;
42.                     endcase
```

下面简单的介绍一下状态机代码：

key[0]作为我们初始化的一个使能信号，如果是实际下板子的时候，我们还需要给按键加一个按键消抖模块。当按键 0 按下之后，代表我们的 SDRAM 的初始化使能信号来了，所以状态机从“IDLE”跳转到了“INIT”状态。在初始化状态，如果我们的初始化模块传来了初始化结束标志“flag_init_end”，那状态机跳转到“ARBIT”仲裁状态，在仲裁状态中，第一个“if”是判断刷新请求的，这也就说明了我们刷新的优先级最高。之后，如果处于仲裁状态，来了读使能信号或者写使能信号并且没有刷新请求，那状态机就跳转到对应的状态。如果处于读或写的状态，当读结束标志或者写结束标志来临的时候（这里的写结束标志和读结束标志都是指突发读或突发写的结束标志），那么就会跳转到仲裁状态。

初始化模块代码简单介绍

下面再简单的看下初始化模块中的代码：

```
1.  /*********************************************
2.  *         Module    Name      :         sdram_init
3.  *         Engineer  :         Kevin
4.  *         Function  :         sdram 初始化模块
5.  *         Date                :         2016.01.10
6.  *         Blog      Website   :         dengkanwen.com
7.  *         Version             :         v1.0
8.  *********************************************/
9.  module    sdram_init(
10.                     input     wire                sclk,               //系统时钟为 50M，即 T=
    20ns
11.                     input     wire                s_rst_n,
12.
13.                     output    reg       [3:0]     cmd_reg,  //sdram 命令寄存器
14.                     output    reg       [11:0]    sdram_addr,         //地址线
15.                     output    reg       [1:0]     sdram_bank,         //bank 地址
16.                     output    reg                 flag_init_end       //sdram 初始化结束标志
17.                     );
18.
19.           parameter CMD_END             =         4'd11,              //初始化结束时的命令计
    数器的值
20.                               CNT_200US =         14'd1_0000,
21.                               NOP               =         4'b0111,  //空操作命令
22.                               PRECHARGE =         4'b0010,  //预充电命令
23.                               AUTO_REF  =         4'b0001,  //自刷新命令
24.                               MRSET             =         4'b0000;  //模式寄存器设置命令
25.
26.           reg       [13:0]    cnt_200us;                    //200us 计数器
27.           reg                 flag_200us;                   //200us 结束标志（200us 结束后，
    一直拉高）
```

```
28.           reg       [3:0]     cnt_cmd;            //命令计数器，便于控制在某个时候发送特定指令
29.           reg                 flag_init;                    //初始化标志：初始化结束后，该标
    志拉低
30. //flag_init
31.           always    @(posedge sclk or negedge s_rst_n)
32.                     if(s_rst_n == 1'b0)
33.                               flag_init <=        1'b1;
34.                     else if(cnt_cmd == CMD_END)
35.                               flag_init <=        1'b0;
36. //cnt_200us
37.           always    @(posedge sclk or negedge s_rst_n)
38.                     if(s_rst_n == 1'b0)
39.                               cnt_200us <=        14'd0;
40.                     else if(cnt_200us == CNT_200US)
41.                               cnt_200us <=        14'd0;
42.                     else if(flag_200us == 1'b0)
43.                               cnt_200us <=        cnt_200us + 1'b1;
44. //flag_200us
45.           always    @(posedge sclk or negedge s_rst_n)
46.                     if(s_rst_n == 1'b0)
47.                               flag_200us          <=        1'b0;
48.                     else if(cnt_200us == CNT_200US)
49.                               flag_200us          <=        1'b1;
50. //cnt_cmd
51.           always    @(posedge sclk or negedge s_rst_n)
52.                     if(s_rst_n == 1'b0)
53.                               cnt_cmd   <=        4'd0;
54.                     else if(flag_200us == 1'b1 && flag_init == 1'b1)
55.                               cnt_cmd   <=        cnt_cmd + 1'b1;
56. //flag_init_end
57.           always    @(posedge sclk or negedge s_rst_n)
58.                     if(s_rst_n == 1'b0)
59.                               flag_init_end       <=        1'b0;
60.                     else if(cnt_cmd == CMD_END)
61.                               flag_init_end       <=        1'b1;
62.                     else
63.                               flag_init_end       <=        1'b0;
64. //cmd_reg
65.           always    @(posedge sclk or negedge s_rst_n)
66.                     if(s_rst_n == 1'b0)
67.                               cmd_reg   <=        NOP;
68.                     else if(cnt_200us == CNT_200US)
69.                               cmd_reg   <=        PRECHARGE;
70.                     else if(flag_200us)
```

```
71.                             case(cnt_cmd)
72.                                     4'd0:
73.                                             cmd_reg   <=        AUTO_REF; //预充电命
    令
74.                                     4'd6:
75.                                             cmd_reg   <=        AUTO_REF;
76.                                     4'd10:
77.                                             cmd_reg   <=        MRSET;
            //模式寄存器设置
78.                                     default:
79.                                             cmd_reg   <=        NOP;
80.                             endcase
81. //sdram_addr
82.           always    @(posedge sclk or negedge s_rst_n)
83.                     if(s_rst_n == 1'b0)
84.                             sdram_addr          <=        12'd0;
85.                     else case(cnt_cmd)
86.                             4'd0:
87.                                     sdram_addr          <=        12'b0100_0000_0000;
            //预充电时，A10 拉高，对所有 Bank 操作
88.                             4'd10:
89.                                     sdram_addr          <=        12'b0000_0011_0010;
            //模式寄存器设置时的指令:CAS=2,Burst Length=4;
90.                             default:
91.                                     sdram_addr          <=        12'd0;
92.                     endcase
93. //sdram_bank
94.           always    @(posedge sclk or negedge s_rst_n)
95.                     if(s_rst_n == 1'b0)
96.                             sdram_bank          <=        2'd0;     //这里仅仅只是初始化，
    在模式寄存器设置时才会用到且其值为全零，故不赋值
97.
98. //sdram_clk
99.           assign    sdram_clk =         ~sclk;
100.
101.          endmodule
```

下面我们来结合代码回顾下初始化过程：

首先，我们需要有 200us 的稳定期，所以我们便有了一个 200us 的计数器 cnt_200us，而这个计数器是根据 flag_200us 的低电平来工作的。大家可以看到，falg_200us 在 200us 计时之后一直拉高。在 200us 计满，即 flag_200us 拉高之后，我们就需要先给一个“NOP”命令，然后给两次“Precharge”命令，同时选中 ALL Banks。

SDRAM 写模块介绍

下面咱们先不对 SDRAM 的初始化进行仿真，等把全部的模块讲完再来仿真。接下来再继续说写操作：首先在我们的仲裁模块，初始化完成之后，就已经跳转到“ARBIT”仲裁状态了，然后，我们在 testbench 中模拟一个外部的写请求信号。咱们先看下写模块的代码：

```
1.  module    sdram_write(
2.                      input     wire               sclk,
3.                      input     wire               s_rst_n,
4.                      input     wire               key_wr,
5.                      input     wire               wr_en,              //来自仲裁模块的写使能
6.                      input     wire               ref_req,  //来自刷新模块的刷新请求
7.                      input     wire      [5:0]    state,              //顶层模块的状态
8.
9.                      output    reg       [15:0]   sdram_dq, //sdram 输入/输出端口
10.                     //output  reg       [3:0]    sdram_dqm,          //输入/输出掩码
11.                     output    reg       [11:0]   sdram_addr,         //sdram 地址线
12.                     output    reg       [1:0]    sdram_bank,         //sdram 的 bank 地址线
13.                     output    reg       [3:0]    sdram_cmd,          //sdram 的命令寄存器
14.                     output    reg                wr_req,             //写请求（不在写状态时向仲裁进行写请求）
15.                     output    reg                flag_wr_end         //写结束标志（有刷新请求来时，向仲裁输出写结束）
16.                     );
17.
18.           parameter NOP       =         4'b0111,  //NOP 命令
19.                               ACT       =         4'b0011,  //ACT 命令
20.                               WR        =         4'b0100,  //写命令（需要将 A10 拉高）
21.                               PRE       =         4'b0010,  //precharge 命令
22.                               CMD_END   =         4'd8,
23.                               COL_END   =         9'd508,             //最后四个列地址的第一个地址
24.                               ROW_END   =         12'd4095, //行地址结束
25.                               AREF      =         6'b10_0000,         //自动刷新状态
26.                               WRITE     =         6'b00_1000;         //状态机的写状态
27.
28.           reg                 flag_act;                     //需要发送 ACT 的标志
29.           reg       [3:0]     cmd_cnt;                      //命令计数器
30.           reg       [11:0]    row_addr;                     //行地址
31.           reg       [11:0]    row_addr_reg;                           //行地址寄存器
32.           reg       [8:0]     col_addr;                     //列地址
```

```
33.           reg                 flag_pre;                   //在 sdram 内部为写状态时需要给 pr
    echarge 命令的标志
34.
35. //flag_pre
36.           always    @(posedge sclk or negedge s_rst_n)
37.                     if(s_rst_n == 1'b0)
38.                               flag_pre  <=        1'b0;
39.                     else if(col_addr == 9'd0 && flag_wr_end == 1'b1)
40.                               flag_pre  <=        1'b1;
41.                     else if(flag_wr_end == 1'b1)
42.                               flag_pre  <=        1'b0;
43.
44. //flag_act
45.           always    @(posedge sclk or negedge s_rst_n)
46.                     if(s_rst_n == 1'b0)
47.                               flag_act  <=        1'b0;
48.                     else if(flag_wr_end)
49.                               flag_act  <=        1'b0;
50.                     else if(ref_req == 1'b1 && state == AREF)
51.                               flag_act  <=        1'b1;
52. //wr_req
53.           always    @(posedge sclk or negedge s_rst_n)
54.                     if(s_rst_n == 1'b0)
55.                               wr_req    <=        1'b0;
56.                     else if(wr_en == 1'b1)
57.                               wr_req    <=        1'b0;
58.                     else if(state != WRITE && key_wr == 1'b1)
59.                               wr_req    <=        1'b1;
60. //flag_wr_end
61.           always    @(posedge sclk or negedge s_rst_n)
62.                     if(s_rst_n == 1'b0)
63.                               flag_wr_end         <=        1'b0;
64.                     else if(cmd_cnt == CMD_END)
65.                               flag_wr_end         <=        1'b1;
66.                     else
67.                               flag_wr_end         <=        1'b0;
68. //cmd_cnt
69.           always    @(posedge sclk or negedge s_rst_n)
70.                     if(s_rst_n == 1'b0)
71.                               cmd_cnt   <=        4'd0;
72.                     else if(state == WRITE)
73.                               cmd_cnt   <=        cmd_cnt + 1'b1;
74.                     else
75.                               cmd_cnt   <=        4'd0;
```

```
76.
77. //sdram_cmd
78.           always    @(posedge sclk or negedge s_rst_n)
79.                     if(s_rst_n == 1'b0)
80.                               sdram_cmd <=        4'd0;
81.                     else case(cmd_cnt)
82.                               3'd1:
83.                                         if(flag_pre == 1'b1)
84.                                                   sdram_cmd <=        PRE;
85.                                         else
86.                                                   sdram_cmd <=        NOP;
87.                               3'd2:
88.                                         if(flag_act == 1'b1 || col_addr == 9'd0)
89.                                                   sdram_cmd <=        ACT;
90.                                         else
91.                                                   sdram_cmd <=        NOP;
92.                               3'd3:
93.                                         sdram_cmd <=        WR;
94.
95.                               default:
96.                                         sdram_cmd <=        NOP;
97.                     endcase
98. //sdram_dq
99.           always    @(posedge sclk or negedge s_rst_n)
100.                              if(s_rst_n == 1'b0)
101.                                        sdram_dq  <=        16'd0;
102.                              else case(cmd_cnt)
103.                                        3'd3:
104.                                                  sdram_dq  <=        16'h0012;
105.                                        3'd4:
106.                                                  sdram_dq  <=        16'h1203;
107.                                        3'd5:
108.                                                  sdram_dq  <=        16'h562f;
109.                                        3'd6:
110.                                                  sdram_dq  <=        16'hfe12;
111.                                        default:
112.                                                  sdram_dq  <=        16'd0;
113.                              endcase
114.          /* //sdram_dq_m
115.                    always    @(posedge sclk or negedge s_rst_n)
116.                              if(s_rst_n == 1'b0)
117.                                        sdram_dqm <=        4'd0; */
118.          //row_addr_reg
119.                    always    @(posedge sclk or negedge s_rst_n)
```

```
                            if(s_rst_n == 1'b0)
                                    row_addr_reg        <=        12'd0;
                            else if(row_addr_reg == ROW_END && col_addr == COL_END && cm
    d_cnt == CMD_END)
                                    row_addr_reg        <=        12'd0;
                            else if(col_addr == COL_END && flag_wr_end == 1'b1)
                                    row_addr_reg        <=        row_addr_reg + 1'b1;

          //row_addr
                    always    @(posedge sclk or negedge s_rst_n)
                            if(s_rst_n == 1'b0)
                                    row_addr  <=        12'd0;
                            else case(cmd_cnt)
                            //因为下边的命令是通过行、列地址分开再给 addr 赋值，所以需要提前一
    个周期赋值，以保证在命令到来时能读到正确的地址
                                    3'd2:
                                            row_addr  <=        12'b0000_0000_0000;
          //在写命令时，不允许 auto-precharge
                                    default:
                                            row_addr  <=        row_addr_reg;
                            endcase
          //col_addr
                    always    @(posedge sclk or negedge s_rst_n)
                            if(s_rst_n == 1'b0)
                                    col_addr  <=        9'd0;
                            else if(col_addr == COL_END && cmd_cnt == CMD_END)
                                    col_addr  <=        9'd0;
                            else if(cmd_cnt == CMD_END)
                                    col_addr  <=        col_addr + 3'd4;
          //sdram_addr
                    always    @(posedge sclk or negedge s_rst_n)
                            if(s_rst_n == 1'b0)
                                    sdram_addr          <=        12'd0;
                            else case(cmd_cnt)
                                    3'd2:
                                            sdram_addr          <=        row_addr;
                                    3'd3:
                                            sdram_addr          <=        col_addr;

                                    default:
                                            sdram_addr          <=        row_addr;
                            endcase
          //sdram_bank
                    always    @(posedge sclk or negedge s_rst_n)
```

```
161.                            if(s_rst_n == 1'b0)
162.                                    sdram_bank          <=        2'b00;
163.          endmodule
```

在我们的模块端口列表中，用 key_wr 来接收写请求信号，这个写请求信号，是在没有写完之前一直拉高的，在写完了全部数据之后才拉低的。当然这个代码的话，还是按照 Kevin 在上一篇博文中的理论来的，大家只要好好理解理论就可以很轻松的明白为什么代码要这样写了。

在写模块中，Kevin 是让 SDRAM 循环着写 16’h0012，16’h1203，16’h562f，16’hfe12 这四个数据。

另外一点，在我们的这个写模块中，Kevin 是在每写完 4 个数据，也就是突发结束后，有一个写完标志，从而使状态机跳转到仲裁状态，然后如果数据没写完，由于写请求是拉高的，所以如果此时没有刷新请求，那状态机还是会跳转到写状态继续写的。

SDRAM 读操作模块

首先，咱们依然先上代码：

```
1.  module    sdram_read(
2.                      input     wire               sclk,
3.                      input     wire               s_rst_n,
4.                      input     wire               rd_en,
5.                      input     wire      [5:0]    state,
6.                      input     wire               ref_req,            //自动刷新请求
7.                      input     wire               key_rd,                       //来自外部
    的读请求信号
8.                      input     wire      [15:0]   rd_dq,              //sdram 的数据端口
9.
10.                     output    reg       [3:0]    sdram_cmd,
11.                     output    reg       [11:0]   sdram_addr,
12.                     output    reg       [1:0]    sdram_bank,
13.                     output    reg                rd_req,                       //读请求
14.                     output    reg                flag_rd_end                   //突发读结
    束标志
15.                     );
16.
17.           parameter NOP       =         4'b0111,
18.                               PRE       =         4'b0010,
19.                               ACT       =         4'b0011,
20.                               RD        =         4'b0101,            //SDRAM 的读命令（给读
    命令时需要给 A10 拉低）
21.                               CMD_END   =         4'd12,                        //
```

```
                                COL_END   =         9'd508,                       //最后四个列地址的第一个地址
                                ROW_END   =         12'd4095,           //行地址结束
                                AREF      =         6'b10_0000,                   //自动刷新状态
                                READ      =         6'b01_0000;                   //状态机的读状态

          reg       [11:0]    row_addr;
          reg       [8:0]     col_addr;
          reg       [3:0]     cmd_cnt;
          reg                 flag_act;                                 //发送 ACT 命令标志（单独设立标志，便于跑高速）

//flag_act
          always    @(posedge sclk or negedge s_rst_n)
                    if(s_rst_n == 1'b0)
                              flag_act  <=        1'b0;
                    else if(flag_rd_end == 1'b1 && ref_req == 1'b1)
                              flag_act  <=        1'b1;
                    else if(flag_rd_end == 1'b1)
                              flag_act  <=        1'b0;
//rd_req
          always    @(posedge sclk or negedge s_rst_n)
                    if(s_rst_n == 1'b0)
                              rd_req    <=        1'b0;
                    else if(rd_en == 1'b1)
                              rd_req    <=        1'b0;
                    else if(key_rd == 1'b1 && state != READ)
                              rd_req    <=        1'b1;
//cmd_cnt
          always    @(posedge sclk or negedge s_rst_n)
                    if(s_rst_n == 1'b0)
                              cmd_cnt   <=        4'd0;
                    else if(state == READ)
                              cmd_cnt   <=        cmd_cnt + 1'b1;
                    else
                              cmd_cnt   <=        4'd0;
//flag_rd_end
          always    @(posedge sclk or negedge s_rst_n)
                    if(s_rst_n == 1'b0)
                              flag_rd_end         <=        1'b0;
                    else if(cmd_cnt == CMD_END)
                              flag_rd_end         <=        1'b1;
```

```
62.                     else
63.                               flag_rd_end         <=        1'b0;
64. //row_addr
65.           always    @(posedge sclk or negedge s_rst_n)
66.                     if(s_rst_n == 1'b0)
67.                               row_addr  <=        12'd0;
68.                     else if(row_addr == ROW_END && col_addr == COL_END && flag_rd_end == 1
    'b1)
69.                               row_addr  <=        12'd0;
70.                     else if(col_addr == COL_END && flag_rd_end == 1'b1)
71.                               row_addr  <=        row_addr + 1'b1;
72. //col_addr
73.           always    @(posedge sclk or negedge s_rst_n)
74.                     if(s_rst_n == 1'b0)
75.                               col_addr  <=        9'd0;
76.                     else if(col_addr == COL_END && flag_rd_end == 1'b1)
77.                               col_addr  <=        9'd0;
78.                     else if(flag_rd_end == 1'b1)
79.                               col_addr  <=        col_addr + 3'd4;
80. //cmd_cnt
81.           always    @(posedge sclk or negedge s_rst_n)
82.                     if(s_rst_n == 1'b0)
83.                               cmd_cnt   <=        4'd0;
84.                     else if(state == READ)
85.                               cmd_cnt   <=        cmd_cnt + 1'b1;
86.                     else
87.                               cmd_cnt   <=        4'd0;
88. //sdram_cmd
89.           always    @(posedge sclk or negedge s_rst_n)
90.                     if(s_rst_n == 1'b0)
91.                               sdram_cmd <=        NOP;
92.                     else case(cmd_cnt)
93.                               4'd2:
94.                                         if(col_addr == 9'd0)
95.                                                   sdram_cmd <=        PRE;
96.                                         else
97.                                                   sdram_cmd <=        NOP;
98.                               4'd3:
99.                                         if(flag_act == 1'b1 || col_addr == 9'd0)
100.                                                  sdram_cmd <=        ACT;
101.                                        else
102.                                                  sdram_cmd <=        NOP;
103.                              4'd4:
104.                                        sdram_cmd <=        RD;
```

```
105.                                    default:
106.                                            sdram_cmd <=        NOP;
107.                            endcase
108.          //sdram_addr
109.                    always    @(posedge sclk or negedge s_rst_n)
110.                              if(s_rst_n == 1'b0)
111.                                      sdram_addr          <=        12'd0;
112.                              else case(cmd_cnt)
113.                                      4'd4:
114.                                              sdram_addr          <=        {3'd0, col
    _addr};
115.                                      default:
116.                                              sdram_addr          <=        row_addr;
117.                              endcase
118.          //sdram_bank
119.                    always    @(posedge sclk or negedge s_rst_n)
120.                              if(s_rst_n == 1'b0)
121.                                      sdram_bank          <=        2'd0;
122.
123.
124.          endmodule
```

其实读模块和写模块是极其相似的，所以在这里，Kevin 就不做赘述，大家慢慢消化吧。

SDRAM 自动刷新模块

自动刷新模块算是比较简单的，等 15us 的时间过了，就向仲裁模块发刷新请求，然后在刷新完成之后，产生刷新结束标志：

```
1.  /*************************************************************
2.  *         Module    Name      :         auto_refresh
3.  *         Enegineer :         Kevin
4.  *         Function  :         sdram 自动刷新
5.  *         Blog      Website   :         http://dengkanwen.com
6.  *         Comment             :         在这个模块中并没有 bank 地址的输出线，需要在顶层模块中设
    置
7.  *************************************************************/
8.  module    auto_refresh(
9.                      input     wire              sclk,
10.                     input     wire              s_rst_n,
11.                     input     wire              ref_en,
12.                     input     wire              flag_init_end,      //初始化结束标志（初始
    化结束后，启动自刷新标志）
```

```
13.
14.                     output    reg       [11:0]    sdram_addr,
15.                     output    reg       [1:0]     sdram_bank,
16.                     output    reg                 ref_req,
17.                     output    reg       [3:0]     cmd_reg,
18.                     output    reg                 flag_ref_end
19.                     );
20.
21.           parameter BANK      =         12'd0100_0000_0000, //自动刷新是对所有 bank 刷新
22.                               CMD_END   =         4'd10,
23.                               CNT_END   =         10'd749,  //15us 计时结束
24.                               NOP       =         4'b0111,  //
25.                               PRE       =         4'b0010,  //precharge 命令
26.                               AREF      =         4'b0001;  //auto-refresh 命令
27.
28.
29.           reg       [9:0]     cnt_15ms; //15ms 计数器
30.           reg                 flag_ref; //处于自刷新阶段标志
31.           reg                 flag_start;         //自动刷新启动标志
32.           reg       [3:0]     cnt_cmd;  //指令计数器
33. //flag_start
34.           always    @(posedge sclk or negedge s_rst_n)
35.                     if(s_rst_n == 1'b0)
36.                               flag_start          <=        1'b0;
37.                     else if(flag_init_end == 1'b1)
38.                               flag_start          <=        1'b1;
39. //cnt_15ms
40.           always    @(posedge sclk or negedge s_rst_n)
41.                     if(s_rst_n == 1'b0)
42.                               cnt_15ms  <=        10'd0;
43.                     else if(cnt_15ms == CNT_END)
44.                               cnt_15ms  <=        10'd0;
45.                     else if(flag_start == 1'b1)
46.                               cnt_15ms  <=        cnt_15ms + 1'b1;
47. //flag_ref
48.           always    @(posedge sclk or negedge s_rst_n)
49.                     if(s_rst_n == 1'b0)
50.                               flag_ref  <=        1'b0;
51.                     else if(cnt_cmd == CMD_END)
52.                               flag_ref  <=        1'b0;
53.                     else if(ref_en == 1'b1)
54.                               flag_ref  <=        1'b1;
55. //cnt_cmd
56.           always    @(posedge sclk or negedge s_rst_n)
```

```
57.                     if(s_rst_n == 1'b0)
58.                               cnt_cmd   <=        4'd0;
59.                     else if(flag_ref == 1'b1)
60.                               cnt_cmd   <=        cnt_cmd + 1'b1;
61.                     else
62.                               cnt_cmd   <=        4'd0;
63. //flag_ref_end
64.           always    @(posedge sclk or negedge s_rst_n)
65.                     if(s_rst_n == 1'b0)
66.                               flag_ref_end        <=        1'b0;
67.                     else if(cnt_cmd == CMD_END)
68.                               flag_ref_end        <=        1'b1;
69.                     else
70.                               flag_ref_end        <=        1'b0;
71. //cmd_reg
72.           always    @(posedge sclk or negedge s_rst_n)
73.                     if(s_rst_n == 1'b0)
74.                               cmd_reg   <=        NOP;
75.                     else case(cnt_cmd)
76.                               3'd0:
77.                                         if(flag_ref == 1'b1)
78.                                                   cmd_reg   <=        PRE;
79.                                         else
80.                                                   cmd_reg   <=        NOP;
81.                               3'd1:
82.                                         cmd_reg   <=        AREF;
83.                               3'd5:
84.                                         cmd_reg   <=        AREF;
85.                               default:
86.                                         cmd_reg   <=        NOP;
87.                     endcase
88. //sdram_addr
89.           always    @(posedge sclk or negedge s_rst_n)
90.                     if(s_rst_n == 1'b0)
91.                               sdram_addr          <=        12'd0;
92.                     else case(cnt_cmd)
93.                               4'd0:
94.                                         sdram_addr          <=        BANK;     //bank 进行
    刷新时指定 allbank or signle bank
95.                               default:
96.                                         sdram_addr          <=        12'd0;
97.                     endcase
98. //sdram_bank
99.           always    @(posedge sclk or negedge s_rst_n)
```

```
100.                            if(s_rst_n == 1'b0)
101.                                    sdram_bank          <=      2'd0;
            //刷新指定的 bank
102.        //ref_req
103.                    always  @(posedge sclk or negedge s_rst_n)
104.                            if(s_rst_n == 1'b0)
105.                                    ref_req   <=        1'b0;
106.                            else if(ref_en == 1'b1)
107.                                    ref_req   <=        1'b0;
108.                            else if(cnt_15ms == CNT_END)
109.                                    ref_req   <=        1'b1;
110.        //flag_ref_end
111.                    always  @(posedge sclk or negedge s_rst_n)
112.                            if(s_rst_n == 1'b0)
113.                                    flag_ref_end        <=      1'b0;
114.                            else if(cnt_cmd == CMD_END)
115.                                    flag_ref_end        <=      1'b1;
116.                            else
117.                                    flag_ref_end        <=      1'b0;
118.
119.         endmodule
```

至此，咱们的整个设计就已经讲完了，至于模块之间怎么连线，Kevin 就不再硬性灌输了，留给大家自己完成吧。当然 Kevin 还是想提醒大家，因为 SDRAM 的数据总线是双向的，所以需要弄个三态门，在向 SDRAM 写数据的时候，模块定义的数据总线应为输出型，在接收数据时，需要定义成高阻态。

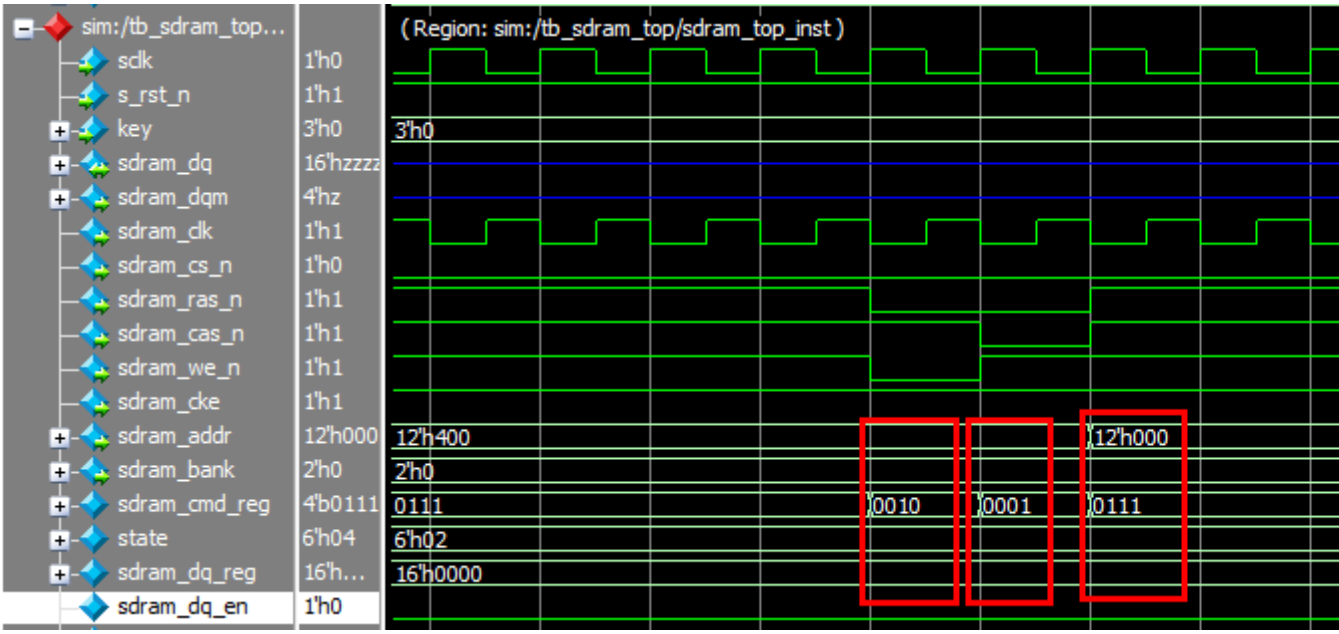
SDRAM 仿真

设计讲完了，咱们来说下仿真，在我们仿真的时候，我们需要用到 SDRAM 的仿真模型（关于仿真模型，Kevin 已经上传到“福利/文档手册”这个栏目下了）。

因为我们需要有一个 200us 的稳定期，所以我们可以先让 Modelsim 跑个 200us，可以直接在命令窗口输入”run 200us”;200us 的稳定器过了之后，接下来应该就是咱们的初始话了，所以我们在让 modelsim 跑 600ns，下面是仿真的结果：

```
VSIM 13> run 200us
VSIM 14> run 600ns
# at time    200150 ns PRE  : Bank = ALL
# at time    200170 ns AREF : Auto Refresh
# at time    200290 ns AREF : Auto Refresh
# at time    200370 ns LMR  : Load Mode Register
#                             CAS Latency      = 3
#                             Burst Length     = 4
#                             Burst Type       = Sequential
#                             Write Burst Mode = Programmed Burst Length

VSIM 15>
```



在上边的仿真中，我们已经知道 SDRAM 已经初始化成功了，设置的潜伏期为 3，突发长度为 4

下面我们再运行一段时间，就运行 1us 吧，往 SDRAM 中写数据：

```
VSIM 13> run 200us
VSIM 14> run 600ns
# at time    200150 ns PRE  : Bank = ALL
# at time    200170 ns AREF : Auto Refresh
# at time    200290 ns AREF : Auto Refresh
# at time    200370 ns LMR  : Load Mode Register
#                             CAS Latency      = 3
#                             Burst Length     = 4
#                             Burst Type       = Sequential
#                             Write Burst Mode = Programmed Burst Length
VSIM 15> run 1us
# at time    200770 ns ACT  : Bank = 0 Row =    0
# at time    200790 ns WRITE: Bank = 0 Row =    0, Col =   0, Data =    18, Dqm = zz00
# at time    200810 ns WRITE: Bank = 0 Row =    0, Col =   1, Data =  4611, Dqm = zz00
# at time    200830 ns WRITE: Bank = 0 Row =    0, Col =   2, Data = 22063, Dqm = zz00
# at time    200850 ns WRITE: Bank = 0 Row =    0, Col =   3, Data = 65042, Dqm = zz00
# at time    201050 ns WRITE: Bank = 0 Row =    0, Col =   4, Data =    18, Dqm = zz00
# at time    201070 ns WRITE: Bank = 0 Row =    0, Col =   5, Data =  4611, Dqm = zz00
# at time    201090 ns WRITE: Bank = 0 Row =    0, Col =   6, Data = 22063, Dqm = zz00
# at time    201110 ns WRITE: Bank = 0 Row =    0, Col =   7, Data = 65042, Dqm = zz00
# at time    201310 ns WRITE: Bank = 0 Row =    0, Col =   8, Data =    18, Dqm = zz00
# at time    201330 ns WRITE: Bank = 0 Row =    0, Col =   9, Data =  4611, Dqm = zz00
# at time    201350 ns WRITE: Bank = 0 Row =    0, Col =  10, Data = 22063, Dqm = zz00
# at time    201370 ns WRITE: Bank = 0 Row =    0, Col =  11, Data = 65042, Dqm = zz00
# at time    201570 ns WRITE: Bank = 0 Row =    0, Col =  12, Data =    18, Dqm = zz00
# at time    201590 ns WRITE: Bank = 0 Row =    0, Col =  13, Data =  4611, Dqm = zz00

VSIM 16>
```

这里的写的数据，就是咱们在写模块中设置的那 4 个数，只是之前的是用 16 进制定义的，这里显示的是 10 进制。

然后我们再看一下读数据：

```
# at time    236210 ns WRITE: Bank = 0 Row =    1, Col =  27, Data = 65042, Dqm = zz00
# at time    236410 ns WRITE: Bank = 0 Row =    1, Col =  28, Data =    18, Dqm = zz00
# at time    236430 ns WRITE: Bank = 0 Row =    1, Col =  29, Data =  4611, Dqm = zz00
# at time    236450 ns WRITE: Bank = 0 Row =    1, Col =  30, Data = 22063, Dqm = zz00
# at time    236470 ns WRITE: Bank = 0 Row =    1, Col =  31, Data = 65042, Dqm = zz00
# at time    236670 ns WRITE: Bank = 0 Row =    1, Col =  32, Data =    18, Dqm = zz00
# at time    236690 ns WRITE: Bank = 0 Row =    1, Col =  33, Data =  4611, Dqm = zz00
# at time    236710 ns WRITE: Bank = 0 Row =    1, Col =  34, Data = 22063, Dqm = zz00
# at time    236730 ns WRITE: Bank = 0 Row =    1, Col =  35, Data = 65042, Dqm = zz00
# at time    236910 ns PRE  : Bank = 0
# at time    236930 ns ACT  : Bank = 0 Row =    0
# at time    236997 ns READ : Bank = 0 Row =    0, Col =   0, Data =    18, Dqm = zz00
# at time    237017 ns READ : Bank = 0 Row =    0, Col =   1, Data =  4611, Dqm = zz00
# at time    237037 ns READ : Bank = 0 Row =    0, Col =   2, Data = 22063, Dqm = zz00
# at time    237057 ns READ : Bank = 0 Row =    0, Col =   3, Data = 65042, Dqm = zz00
# at time    237337 ns READ : Bank = 0 Row =    0, Col =   4, Data =    18, Dqm = zz00
# at time    237357 ns READ : Bank = 0 Row =    0, Col =   5, Data =  4611, Dqm = zz00
# at time    237377 ns READ : Bank = 0 Row =    0, Col =   6, Data = 22063, Dqm = zz00
# at time    237397 ns READ : Bank = 0 Row =    0, Col =   7, Data = 65042, Dqm = zz00
# at time    237677 ns READ : Bank = 0 Row =    0, Col =   8, Data =    18, Dqm = zz00
# at time    237697 ns READ : Bank = 0 Row =    0, Col =   9, Data =  4611, Dqm = zz00
# at time    237717 ns READ : Bank = 0 Row =    0, Col =  10, Data = 22063, Dqm = zz00
# at time    237737 ns READ : Bank = 0 Row =    0, Col =  11, Data = 65042, Dqm = zz00
# at time    238017 ns READ : Bank = 0 Row =    0, Col =  12, Data =    18, Dqm = zz00
# at time    238037 ns READ : Bank = 0 Row =    0, Col =  13, Data =  4611, Dqm = zz00
# at time    238057 ns READ : Bank = 0 Row =    0, Col =  14, Data = 22063, Dqm = zz00
```

大家可以看下，我们读出来的数据和写进来的数据是不是一样的呢？