[SDRAM驱动篇之简易SDRAM控制器的verilog代码实现](http://dengkanwen.com/137.html)

[FPGA学习笔记](http://dengkanwen.com/notes) [Kevin](http://dengkanwen.com/author/kevin) 2年前 (2016-01-17) 8899℃ [0评论](http://dengkanwen.com/137.html#comments)

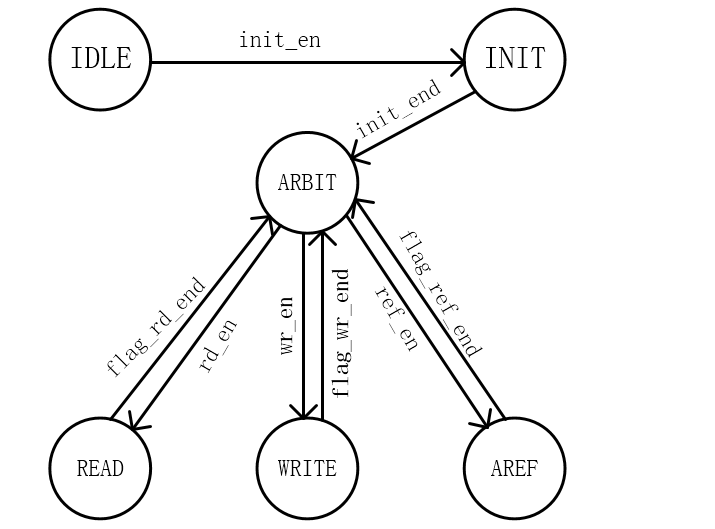
在Kevin写的上一篇博文《[SDRAM理论篇之基础知识及操作时序](http://dengkanwen.com/105.html" \o ")》中，已经把SDRAM工作的基本原理和SDRAM初始化、读、写及自动刷新操作的时序讲清楚了，在这一片博文中，Kevin来根据在上一篇博文中分析的思路来把写一个简单的SDRAM控制器。

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。

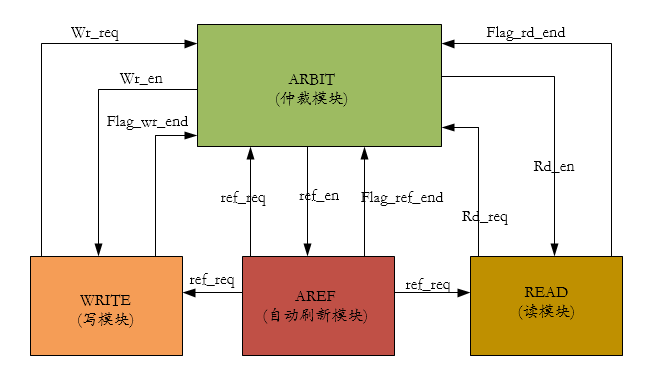
大家看完可能会想，说是这么说，那代码怎么来写呢？似乎还是没什么思路。大家可以想象一下，我们写的SDRAM控制器是肯定包括初始化、读操作、写操作及自动刷新这些操作的，既然这样，我们就可以给每一个操作写上一个模块独立开来，这样也便于我们每个模块的调试，显然这种思路是正确的。那怎么让我们的各个模块工作起来呢，虽然都是独立的模块，但很显然这几个模块之间又是相互关联的。就拿上面刚才说的那个情况来讲，如果SDRAM需要刷新了，而SDRAM却正在执行写操作，那我们刷新模块与写模块之间怎么进行控制呢？这个问题解决了，读模块与刷新模块之间的这个问题也可以很轻松的解决。大家不妨可以自己先想一下。

**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：

[](http://dengkanwen.com/wp-content/uploads/2016/01/sdram_state-1.png)

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：

[](http://dengkanwen.com/wp-content/uploads/2016/01/state_module.png)

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,
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. );
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
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
98. //sdram\_clk
99. assign sdram\_clk = ~sclk;
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
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. );
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
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precharge命令的标志
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;
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;
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;
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)
120. if(s\_rst\_n == 1'b0)
121. row\_addr\_reg <= 12'd0;
122. else if(row\_addr\_reg == ROW\_END && col\_addr == COL\_END && cmd\_cnt == CMD\_END)
123. row\_addr\_reg <= 12'd0;
124. else if(col\_addr == COL\_END && flag\_wr\_end == 1'b1)
125. row\_addr\_reg <= row\_addr\_reg + 1'b1;
127. //row\_addr
128. always @(posedge sclk or negedge s\_rst\_n)
129. if(s\_rst\_n == 1'b0)
130. row\_addr <= 12'd0;
131. else case(cmd\_cnt)
132. //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
133. 3'd2:
134. row\_addr <= 12'b0000\_0000\_0000; //在写命令时，不允许auto-precharge
135. default:
136. row\_addr <= row\_addr\_reg;
137. endcase
138. //col\_addr
139. always @(posedge sclk or negedge s\_rst\_n)
140. if(s\_rst\_n == 1'b0)
141. col\_addr <= 9'd0;
142. else if(col\_addr == COL\_END && cmd\_cnt == CMD\_END)
143. col\_addr <= 9'd0;
144. else if(cmd\_cnt == CMD\_END)
145. col\_addr <= col\_addr + 3'd4;
146. //sdram\_addr
147. always @(posedge sclk or negedge s\_rst\_n)
148. if(s\_rst\_n == 1'b0)
149. sdram\_addr <= 12'd0;
150. else case(cmd\_cnt)
151. 3'd2:
152. sdram\_addr <= row\_addr;
153. 3'd3:
154. sdram\_addr <= col\_addr;
156. default:
157. sdram\_addr <= row\_addr;
158. endcase
159. //sdram\_bank
160. 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
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. );
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, //
22. COL\_END = 9'd508, //最后四个列地址的第一个地址
23. ROW\_END = 12'd4095, //行地址结束
24. AREF = 6'b10\_0000, //自动刷新状态
25. READ = 6'b01\_0000; //状态机的读状态
27. reg [11:0] row\_addr;
28. reg [8:0] col\_addr;
29. reg [3:0] cmd\_cnt;
30. reg flag\_act; //发送ACT命令标志（单独设立标志，便于跑高速）
32. //flag\_act
33. always @(posedge sclk or negedge s\_rst\_n)
34. if(s\_rst\_n == 1'b0)
35. flag\_act <= 1'b0;
36. else if(flag\_rd\_end == 1'b1 && ref\_req == 1'b1)
37. flag\_act <= 1'b1;
38. else if(flag\_rd\_end == 1'b1)
39. flag\_act <= 1'b0;
40. //rd\_req
41. always @(posedge sclk or negedge s\_rst\_n)
42. if(s\_rst\_n == 1'b0)
43. rd\_req <= 1'b0;
44. else if(rd\_en == 1'b1)
45. rd\_req <= 1'b0;
46. else if(key\_rd == 1'b1 && state != READ)
47. rd\_req <= 1'b1;
48. //cmd\_cnt
49. always @(posedge sclk or negedge s\_rst\_n)
50. if(s\_rst\_n == 1'b0)
51. cmd\_cnt <= 4'd0;
52. else if(state == READ)
53. cmd\_cnt <= cmd\_cnt + 1'b1;
54. else
55. cmd\_cnt <= 4'd0;
56. //flag\_rd\_end
57. always @(posedge sclk or negedge s\_rst\_n)
58. if(s\_rst\_n == 1'b0)
59. flag\_rd\_end <= 1'b0;
60. else if(cmd\_cnt == CMD\_END)
61. 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;

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）
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. );
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

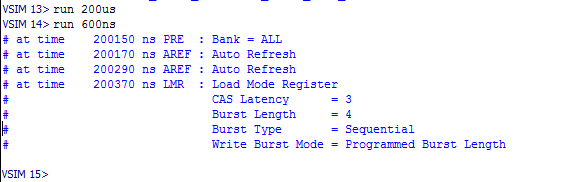
29. reg [9:0] cnt\_15ms; //15ms计数器
30. reg flag\_ref; //处于自刷新阶段标志
31. reg flag\_start; //自动刷新启动标志
32. reg [3:0] cnt\_cmd; //指令计数器
33. //flag\_start
34. always @(posedge sclk or negedge s\_rst\_n)
35. if(s\_rst\_n == 1'b0)
36. flag\_start <= 1'b0;
37. else if(flag\_init\_end == 1'b1)
38. flag\_start <= 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;
119. endmodule

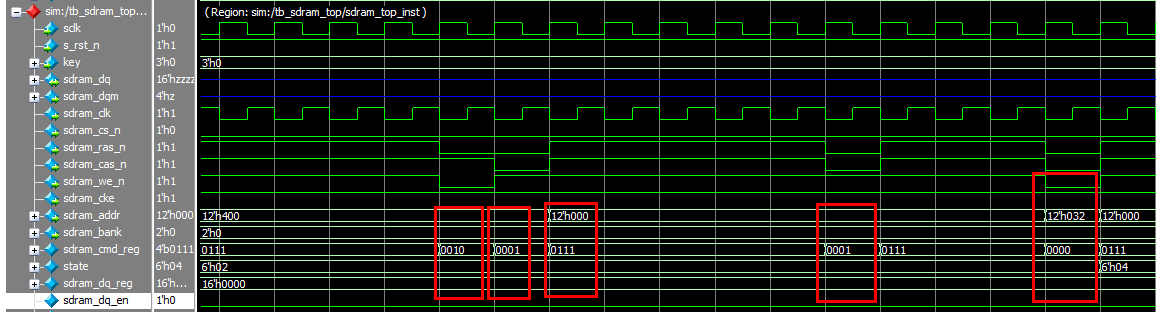
至此，咱们的整个设计就已经讲完了，至于模块之间怎么连线，Kevin就不再硬性灌输了，留给大家自己完成吧。当然Kevin还是想提醒大家，因为SDRAM的数据总线是双向的，所以需要弄个三态门，在向SDRAM写数据的时候，模块定义的数据总线应为输出型，在接收数据时，需要定义成高阻态。

**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）。

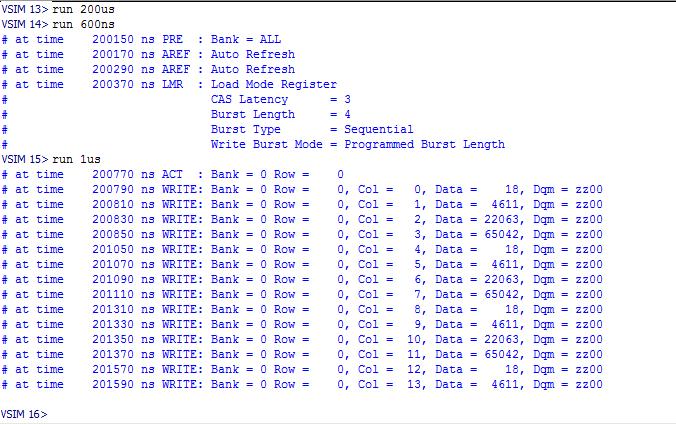
因为我们需要有一个200us的稳定期，所以我们可以先让Modelsim跑个200us，可以直接在命令窗口输入”run 200us”;200us的稳定器过了之后，接下来应该就是咱们的初始话了，所以我们在让modelsim跑600ns，下面是仿真的结果：

[](http://dengkanwen.com/wp-content/uploads/2016/01/sdram_c.png)

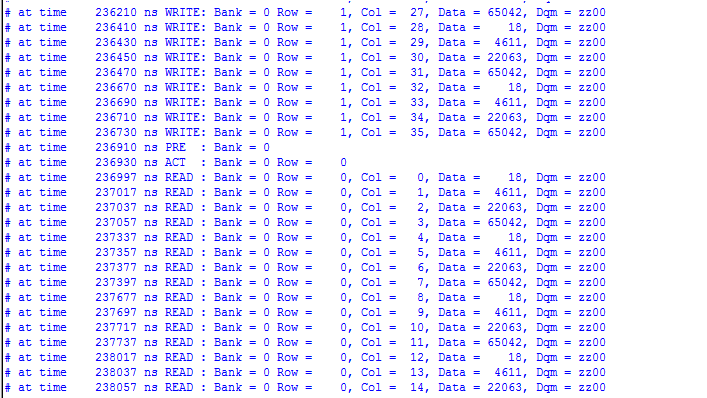
[](http://dengkanwen.com/wp-content/uploads/2016/01/sdram_tb_wave.png)

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：

[](http://dengkanwen.com/wp-content/uploads/2016/01/sdram_wave.png)

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：[](http://dengkanwen.com/wp-content/uploads/2016/01/sdram_read.png)

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？