

```

1  /**
2   * simple controller for ISSI IS42S16160G-7 SDRAM found in De0 Nano
3   * 16Mbit x 16 data bit bus (32 megabytes)
4   * Default options
5   *   133Mhz
6   *   CAS 3
7   *
8   * Very simple host interface
9   *   * No burst support
10  *   * haddr - address for reading and wring 16 bits of data
11  *   * data_input - data for writing, latched in when wr_enable is highz0
12  *   * data_output - data for reading, comes available sometime
13  *   *few clocks* after rd_enable and address is presented on bus
14  *   * rst_n - start init ram process
15  *   * rd_enable - read enable, on clk posedge haddr will be latched in,
16  *   after *few clocks* data will be available on the data_output port
17  *   * wr_enable - write enable, on clk posedge haddr and data_input will
18  *   be latched in, after *few clocks* data will be written to sdram
19  *
20  * Theory
21  * This simple host interface has a busy signal to tell you when you are
22  * not able to issue commands.
23  */
24
25 module sdram_controller (
26     /* HOST INTERFACE */
27     wr_addr,
28     wr_data,
29     wr_enable,
30
31     rd_addr,
32     rd_data,
33     rd_ready,
34     rd_enable,
35
36     busy, rst_n, clk,
37
38     /* SDRAM SIDE */
39     addr, bank_addr, data, clock_enable, cs_n, ras_n, cas_n, we_n,
40     data_mask_low, data_mask_high
41 );
42
43 /* Internal Parameters */
44 parameter ROW_WIDTH = 9;
45 parameter COL_WIDTH = 9;
46 parameter BANK_WIDTH = 2;
47
48 parameter SDRADDR_WIDTH = ROW_WIDTH > COL_WIDTH ? ROW_WIDTH : COL_WIDTH;
49 parameter HADDR_WIDTH = BANK_WIDTH + ROW_WIDTH + COL_WIDTH;
50
51 parameter CLK_FREQUENCY = 25; // Mhz
52 parameter REFRESH_TIME = 32; // ms (how often we need to refresh)
53 parameter REFRESH_COUNT = 8192; // cycles (how many refreshes required per refresh time)
54
55 // clk / refresh = clk / sec
56 // , sec / refbatch
57 // , ref / refbatch
58 localparam CYCLES_BETWEEN_REFRESH = ( CLK_FREQUENCY
59     * 1_000
60     * REFRESH_TIME
61     ) / REFRESH_COUNT;
62
63 // STATES - State
64 localparam IDLE = 5'b00000;
65
66 localparam INIT_NOP1 = 5'b01000,
67     INIT_PRE1 = 5'b01001,
68     INIT_NOP1_1 = 5'b00101,
69     INIT_REF1 = 5'b01010,
70     INIT_NOP2 = 5'b01011,

```

```

71         INIT_REF2 = 5'b01100 ,
72         INIT_NOP3 = 5'b01101 ,
73         INIT_LOAD = 5'b01110 ,
74         INIT_NOP4 = 5'b01111 ;
75
76     localparam REF_PRE = 5'b00001 ,
77         REF_NOP1 = 5'b00010 ,
78         REF_REF = 5'b00011 ,
79         REF_NOP2 = 5'b00100 ;
80
81     localparam READ_ACT = 5'b10000 ,
82         READ_NOP1 = 5'b10001 ,
83         READ_CAS = 5'b10010 ,
84         READ_NOP2 = 5'b10011 ,
85         READ_READ = 5'b10100 ;
86
87     localparam WRIT_ACT = 5'b11000 ,
88         WRIT_NOP1 = 5'b11001 ,
89         WRIT_CAS = 5'b11010 ,
90         WRIT_NOP2 = 5'b11011 ;
91
92     // Commands          CCRCWBBA
93     //                   ESSSE100
94     localparam CMD_PALL = 8'b10010001 ,
95         CMD_REF = 8'b10001000 ,
96         CMD_NOP = 8'b10111000 ,
97         CMD_MRS = 8'b1000000x ,
98         CMD_BACT = 8'b10011xxx ,
99         CMD_READ = 8'b10101xx1 ,
100        CMD_WRIT = 8'b10100xx1 ;
101
102     /* Interface Definition */
103     /* HOST INTERFACE */
104     input  [HADDR_WIDTH-1:0]  wr_addr ;
105     input  [15:0]             wr_data ;
106     input                      wr_enable ;
107
108     input  [HADDR_WIDTH-1:0]  rd_addr ;
109     output [15:0]             rd_data ;
110     input                    rd_enable ;
111     output                   rd_ready ;
112
113     output                   busy ;
114     input                   rst_n ;
115     input                   clk ;
116
117     /* SDRAM SIDE */
118     output [SDRADDR_WIDTH-1:0] addr ;
119     output [BANK_WIDTH-1:0]    bank_addr ;
120     inout  [15:0]              data ;
121     output                   clock_enable ;
122     output                   cs_n ;
123     output                   ras_n ;
124     output                   cas_n ;
125     output                   we_n ;
126     output                   data_mask_low ;
127     output                   data_mask_high ;
128
129     /* I/O Registers */
130
131     reg  [HADDR_WIDTH-1:0] haddr_r ;
132     reg  [15:0]           wr_data_r ;
133     reg  [15:0]           rd_data_r ;
134     reg                   busy ;
135     reg                   data_mask_low_r ;
136     reg                   data_mask_high_r ;
137     reg [SDRADDR_WIDTH-1:0] addr_r ;
138     reg [BANK_WIDTH-1:0]    bank_addr_r ;
139     reg                   rd_ready_r ;
140

```

```

141 wire [15:0]          data_output;
142 wire                data_mask_low, data_mask_high;
143
144 assign data_mask_high = data_mask_high_r;
145 assign data_mask_low  = data_mask_low_r;
146 assign rd_data        = rd_data_r;
147
148 /* Internal wiring */
149 reg [3:0] state_cnt;
150 reg [9:0] refresh_cnt;
151
152 reg [7:0] command;
153 reg [4:0] state;
154
155 // TODO output addr[6:4] when programming mode register
156
157 reg [7:0] command_nxt;
158 reg [3:0] state_cnt_nxt;
159 reg [4:0] next;
160
161 assign {clock_enable, cs_n, ras_n, cas_n, we_n} = command[7:3];
162 // state[4] will be set if mode is read/write
163 assign bank_addr      = (state[4]) ? bank_addr_r : command[2:1];
164 assign addr           = (state[4] | state == INIT_LOAD) ? addr_r : {
{SDRADDR_WIDTH-11{1'b0}}, command[0], 10'd0 };
165
166 assign data = (state == WRIT_CAS) ? wr_data_r : 16'bz;
167 assign rd_ready = rd_ready_r;
168
169 // HOST INTERFACE
170 // all registered on posedge
171 always @ (posedge clk)
172   if (~rst_n)
173     begin
174       state <= INIT_NOP1;
175       command <= CMD_NOP;
176       state_cnt <= 4'hf;
177
178       haddr_r <= {HADDR_WIDTH{1'b0}};
179       wr_data_r <= 16'b0;
180       rd_data_r <= 16'b0;
181       busy <= 1'b0;
182     end
183   else
184     begin
185
186       state <= next;
187       command <= command_nxt;
188
189       if (!state_cnt)
190         state_cnt <= state_cnt_nxt;
191       else
192         state_cnt <= state_cnt - 1'b1;
193
194       if (wr_enable)
195         wr_data_r <= wr_data;
196
197       if (state == READ_READ)
198         begin
199           rd_data_r <= data;
200           rd_ready_r <= 1'b1;
201         end
202       else
203         rd_ready_r <= 1'b0;
204
205       busy <= state[4];
206
207       if (rd_enable)
208         haddr_r <= rd_addr;
209       else if (wr_enable)

```

```

210         haddr_r <= wr_addr;
211
212     end
213
214     // Handle refresh counter
215     always @ (posedge clk)
216     if (~rst_n)
217         refresh_cnt <= 10'b0;
218     else
219         if (state == REF_NOP2)
220             refresh_cnt <= 10'b0;
221         else
222             refresh_cnt <= refresh_cnt + 1'b1;
223
224
225     /* Handle logic for sending addresses to SDRAM based on current state*/
226     always @*
227     begin
228         if (state[4])
229             {data_mask_low_r, data_mask_high_r} = 2'b00;
230         else
231             {data_mask_low_r, data_mask_high_r} = 2'b11;
232
233         bank_addr_r = 2'b00;
234         addr_r = {SDRADDR_WIDTH{1'b0}};
235
236         if (state == READ_ACT | state == WRIT_ACT)
237             begin
238                 bank_addr_r = haddr_r[HADDR_WIDTH-1:HADDR_WIDTH-(BANK_WIDTH)];
239                 addr_r = haddr_r[HADDR_WIDTH-(BANK_WIDTH+1):HADDR_WIDTH-(BANK_WIDTH+ROW_WIDTH)];
240             end
241         else if (state == READ_CAS | state == WRIT_CAS)
242             begin
243                 // Send Column Address
244                 // Set bank to bank to precharge
245                 bank_addr_r = haddr_r[HADDR_WIDTH-1:HADDR_WIDTH-(BANK_WIDTH)];
246
247                 // Examples for math
248                 //          BANK   ROW    COL
249                 // HADDR_WIDTH  2 +   13 +   9   = 24
250                 // SDRADDR_WIDTH 13
251
252                 // Set CAS address to:
253                 //   0s,
254                 //   1 (A10 is always for auto precharge),
255                 //   0s,
256                 //   column address
257                 addr_r = {
258                     {SDRADDR_WIDTH-(11){1'b0}},
259                     1'b1,                               /* A10 */
260                     {10-COL_WIDTH{1'b0}},
261                     haddr_r[COL_WIDTH-1:0]
262                 };
263             end
264         else if (state == INIT_LOAD)
265             begin
266                 // Program mode register during load cycle
267                 //
268                 //          B   C   SB
269                 //          R   A   EUR
270                 //          S   S-3Q ST
271                 //          T   654L210
272                 addr_r = {{SDRADDR_WIDTH-10{1'b0}}, 10'b1000110000};
273             end
274     end
275
276     // Next state logic
277     always @*
278     begin
279         state_cnt_nxt = 4'd0;
280         command_nxt = CMD_NOP;

```

```
280     if (state == IDLE)
281         // Monitor for refresh or hold
282         if (refresh_cnt >= CYCLES_BETWEEN_REFRESH)
283             begin
284                 next = REF_PRE;
285                 command_nxt = CMD_PALL;
286             end
287         else if (rd_enable)
288             begin
289                 next = READ_ACT;
290                 command_nxt = CMD_BACT;
291             end
292         else if (wr_enable)
293             begin
294                 next = WRIT_ACT;
295                 command_nxt = CMD_BACT;
296             end
297         else
298             begin
299                 // HOLD
300                 next = IDLE;
301             end
302     else
303         if (!state_cnt)
304             case (state)
305                 // INIT ENGINE
306                 INIT_NOP1:
307                     begin
308                         next = INIT_PRE1;
309                         command_nxt = CMD_PALL;
310                     end
311                 INIT_PRE1:
312                     begin
313                         next = INIT_NOP1_1;
314                     end
315                 INIT_NOP1_1:
316                     begin
317                         next = INIT_REF1;
318                         command_nxt = CMD_REF;
319                     end
320                 INIT_REF1:
321                     begin
322                         next = INIT_NOP2;
323                         state_cnt_nxt = 4'd7;
324                     end
325                 INIT_NOP2:
326                     begin
327                         next = INIT_REF2;
328                         command_nxt = CMD_REF;
329                     end
330                 INIT_REF2:
331                     begin
332                         next = INIT_NOP3;
333                         state_cnt_nxt = 4'd7;
334                     end
335                 INIT_NOP3:
336                     begin
337                         next = INIT_LOAD;
338                         command_nxt = CMD_MRS;
339                     end
340                 INIT_LOAD:
341                     begin
342                         next = INIT_NOP4;
343                         state_cnt_nxt = 4'd1;
344                     end
345                 // INIT_NOP4: default - IDLE
346
347                 // REFRESH
348                 REF_PRE:
349                     begin
```

```
350         next = REF_NOP1;
351     end
352 REF_NOP1:
353     begin
354         next = REF_REF;
355         command_nxt = CMD_REF;
356     end
357 REF_REF:
358     begin
359         next = REF_NOP2;
360         state_cnt_nxt = 4'd7;
361     end
362 // REF_NOP2: default - IDLE
363
364 // WRITE
365 WRIT_ACT:
366     begin
367         next = WRIT_NOP1;
368         state_cnt_nxt = 4'd1;
369     end
370 WRIT_NOP1:
371     begin
372         next = WRIT_CAS;
373         command_nxt = CMD_WRIT;
374     end
375 WRIT_CAS:
376     begin
377         next = WRIT_NOP2;
378         state_cnt_nxt = 4'd1;
379     end
380 // WRIT_NOP2: default - IDLE
381
382 // READ
383 READ_ACT:
384     begin
385         next = READ_NOP1;
386         state_cnt_nxt = 4'd1;
387     end
388 READ_NOP1:
389     begin
390         next = READ_CAS;
391         command_nxt = CMD_READ;
392     end
393 READ_CAS:
394     begin
395         next = READ_NOP2;
396         state_cnt_nxt = 4'd1;
397     end
398 READ_NOP2:
399     begin
400         next = READ_READ;
401     end
402 // READ_READ: default - IDLE
403
404 default:
405     begin
406         next = IDLE;
407     end
408 endcase
409 else
410     begin
411         // Counter Not Reached - HOLD
412         next = state;
413         command_nxt = command;
414     end
415 end
416 endmodule
417
418
```