```verilog
/**
 * simple controller for ISSI IS42S16160G-7 SDRAM found in De0 Nano
 *  16Mbit x 16 data bit bus (32 megabytes)
 *  Default options
 *    133Mhz
 *    CAS 3
 *
 *  Very simple host interface
 *     * No burst support
 *     * haddr - address for reading and wriging 16 bits of data
 *     * data_input - data for writing, latched in when wr_enable is highz0
 *     * data_output - data for reading, comes available sometime
 *       *few clocks* after rd_enable and address is presented on bus
 *     * rst_n - start init ram process
 *     * rd_enable - read enable, on clk posedge haddr will be latched in,
 *       after *few clocks* data will be available on the data_output port
 *     * wr_enable - write enable, on clk posedge haddr and data_input will
 *       be latched in, after *few clocks* data will be written to sdram
 *
 * Theory
 *  This simple host interface has a busy signal to tell you when you are
 *  not able to issue commands.
 */

module sdram_controller (
    /* HOST INTERFACE */
    wr_addr,
    wr_data,
    wr_enable,

    rd_addr,
    rd_data,
    rd_ready,
    rd_enable,

    busy, rst_n, clk,

    /* SDRAM SIDE */
    addr, bank_addr, data, clock_enable, cs_n, ras_n, cas_n, we_n,
    data_mask_low, data_mask_high
);

/* Internal Parameters */
parameter ROW_WIDTH = 9;
parameter COL_WIDTH = 9;
parameter BANK_WIDTH = 2;

parameter SDRADDR_WIDTH = ROW_WIDTH > COL_WIDTH ? ROW_WIDTH : COL_WIDTH;
parameter HADDR_WIDTH = BANK_WIDTH + ROW_WIDTH + COL_WIDTH;

parameter CLK_FREQUENCY = 25;  // Mhz
parameter REFRESH_TIME =  32;   // ms     (how often we need to refresh)
parameter REFRESH_COUNT = 8192; // cycles (how many refreshes required per refresh time)

// clk / refresh =  clk / sec
//                , sec / refbatch
//                , ref / refbatch
localparam CYCLES_BETWEEN_REFRESH = ( CLK_FREQUENCY
                                      * 1_000
                                      * REFRESH_TIME
                                    ) / REFRESH_COUNT;

// STATES - State
localparam IDLE      = 5'b00000;

localparam INIT_NOP1 = 5'b01000,
           INIT_PRE1 = 5'b01001,
           INIT_NOP1_1=5'b00101,
           INIT_REF1 = 5'b01010,
           INIT_NOP2 = 5'b01011,
           INIT_REF2  = 5'b01100,
           INIT_NOP3  = 5'b01101,
           INIT_LOAD  = 5'b01110,
           INIT_NOP4  = 5'b01111;

localparam REF_PRE   =  5'b00001,
           REF_NOP1  =  5'b00010,
           REF_REF   =  5'b00011,
           REF_NOP2  =  5'b00100;

localparam READ_ACT   = 5'b10000,
           READ_NOP1  = 5'b10001,
           READ_CAS   = 5'b10010,
           READ_NOP2  = 5'b10011,
           READ_READ  = 5'b10100;

localparam WRIT_ACT   = 5'b11000,
           WRIT_NOP1  = 5'b11001,
           WRIT_CAS   = 5'b11010,
           WRIT_NOP2  = 5'b11011;

// Commands              CCRCWBBA
//                       ESSSE100
localparam CMD_PALL  = 8'b10010001,
           CMD_REF   = 8'b10001000,
           CMD_NOP   = 8'b10111000,
           CMD_MRS   = 8'b1000000x,
           CMD_BACT  = 8'b10011xxx,
           CMD_READ  = 8'b10101xx1,
           CMD_WRIT  = 8'b10100xx1;

/* Interface Definition */
/* HOST INTERFACE */
input  [HADDR_WIDTH-1:0]    wr_addr;
input  [15:0]               wr_data;
input                       wr_enable;

input  [HADDR_WIDTH-1:0]    rd_addr;
output [15:0]               rd_data;
input                       rd_enable;
output                      rd_ready;

output                      busy;
input                       rst_n;
input                       clk;

/* SDRAM SIDE */
output [SDRADDR_WIDTH-1:0] addr;
output [BANK_WIDTH-1:0]    bank_addr;
inout  [15:0]              data;
output                     clock_enable;
output                     cs_n;
output                     ras_n;
output                     cas_n;
output                     we_n;
output                     data_mask_low;
output                     data_mask_high;

/* I/O Registers */

reg  [HADDR_WIDTH-1:0]   haddr_r;
reg  [15:0]              wr_data_r;
reg  [15:0]              rd_data_r;
reg                      busy;
reg                      data_mask_low_r;
reg                      data_mask_high_r;
reg [SDRADDR_WIDTH-1:0]  addr_r;
reg [BANK_WIDTH-1:0]     bank_addr_r;
reg                      rd_ready_r;

```

```verilog
wire [15:0]              data_output;
wire                     data_mask_low, data_mask_high;

assign data_mask_high = data_mask_high_r;
assign data_mask_low  = data_mask_low_r;
assign rd_data        = rd_data_r;

/* Internal Wiring */
reg [3:0] state_cnt;
reg [9:0] refresh_cnt;

reg [7:0] command;
reg [4:0] state;

// TODO output addr[6:4] when programming mode register

reg [7:0] command_nxt;
reg [3:0] state_cnt_nxt;
reg [4:0] next;

assign {clock_enable, cs_n, ras_n, cas_n, we_n} = command[7:3];
// state[4] will be set if mode is read/write
assign bank_addr      = (state[4]) ? bank_addr_r : command[2:1];
assign addr           = (state[4] | state == INIT_LOAD) ? addr_r : {
{SDRADDR_WIDTH-11{1'b0}}, command[0], 10'd0 };

assign data = (state == WRIT_CAS) ? wr_data_r : 16'bz;
assign rd_ready = rd_ready_r;

// HOST INTERFACE
// all registered on posedge
always @ (posedge clk)
  if (~rst_n)
    begin
    state <= INIT_NOP1;
    command <= CMD_NOP;
    state_cnt <= 4'hf;

    haddr_r <= {HADDR_WIDTH{1'b0}};
    wr_data_r <= 16'b0;
    rd_data_r <= 16'b0;
    busy <= 1'b0;
    end
  else
    begin

    state <= next;
    command <= command_nxt;

    if (!state_cnt)
      state_cnt <= state_cnt_nxt;
    else
      state_cnt <= state_cnt - 1'b1;

    if (wr_enable)
      wr_data_r <= wr_data;

    if (state == READ_READ)
      begin
      rd_data_r <= data;
      rd_ready_r <= 1'b1;
      end
    else
      rd_ready_r <= 1'b0;

    busy <= state[4];

    if (rd_enable)
      haddr_r <= rd_addr;
    else if (wr_enable)
```

```
         haddr_r <= wr_addr;

       end

   // Handle refresh counter
   always @ (posedge clk)
    if (~rst_n)
      refresh_cnt <= 10'b0;
    else
      if (state == REF_NOP2)
        refresh_cnt <= 10'b0;
      else
        refresh_cnt <= refresh_cnt + 1'b1;


   /* Handle logic for sending addresses to SDRAM based on current state*/
   always @*
   begin
       if (state[4])
         {data_mask_low_r, data_mask_high_r} = 2'b00;
       else
         {data_mask_low_r, data_mask_high_r} = 2'b11;

      bank_addr_r = 2'b00;
      addr_r = {SDRADDR_WIDTH{1'b0}};

      if (state == READ_ACT | state == WRIT_ACT)
        begin
        bank_addr_r = haddr_r[HADDR_WIDTH-1:HADDR_WIDTH-(BANK_WIDTH)];
        addr_r = haddr_r[HADDR_WIDTH-(BANK_WIDTH+1):HADDR_WIDTH-(BANK_WIDTH+ROW_WIDTH)];
        end
      else if (state == READ_CAS | state == WRIT_CAS)
        begin
        // Send Column Address
        // Set bank to bank to precharge
        bank_addr_r = haddr_r[HADDR_WIDTH-1:HADDR_WIDTH-(BANK_WIDTH)];

        // Examples for math
        //               BANK  ROW    COL
        // HADDR_WIDTH   2 +   13 +   9   = 24
        // SDRADDR_WIDTH 13

        // Set CAS address to:
        //   0s,
        //   1 (A10 is always for auto precharge),
        //   0s,
        //   column address
        addr_r = {
                   {SDRADDR_WIDTH-(11){1'b0}},
                   1'b1,                        /* A10 */
                   {10-COL_WIDTH{1'b0}},
                   haddr_r[COL_WIDTH-1:0]
                 };
        end
      else if (state == INIT_LOAD)
        begin
        // Program mode register during load cycle
        //                                    B  C  SB
        //                                    R  A  EUR
        //                                    S  S-3Q ST
        //                                    T  654L210
        addr_r = {{SDRADDR_WIDTH-10{1'b0}}, 10'b1000110000};
        end
   end

   // Next state logic
   always @*
   begin
      state_cnt_nxt = 4'd0;
      command_nxt = CMD_NOP;
```

```verilog
      if (state == IDLE)
          // Monitor for refresh or hold
          if (refresh_cnt >= CYCLES_BETWEEN_REFRESH )
            begin
            next = REF_PRE;
            command_nxt = CMD_PALL;
            end
          else if (rd_enable)
            begin
            next = READ_ACT;
            command_nxt = CMD_BACT;
            end
          else if (wr_enable)
            begin
            next = WRIT_ACT;
            command_nxt = CMD_BACT;
            end
          else
            begin
            // HOLD
            next = IDLE;
            end
       else
         if (!state_cnt)
          case (state)
            // INIT ENGINE
            INIT_NOP1:
              begin
              next = INIT_PRE1;
              command_nxt = CMD_PALL;
              end
            INIT_PRE1:
              begin
              next = INIT_NOP1_1;
              end
            INIT_NOP1_1:
              begin
              next = INIT_REF1;
              command_nxt = CMD_REF;
              end
            INIT_REF1:
              begin
              next = INIT_NOP2;
              state_cnt_nxt = 4'd7;
              end
            INIT_NOP2:
              begin
              next = INIT_REF2;
              command_nxt = CMD_REF;
              end
            INIT_REF2:
              begin
              next = INIT_NOP3;
              state_cnt_nxt = 4'd7;
              end
            INIT_NOP3:
              begin
              next = INIT_LOAD;
              command_nxt = CMD_MRS;
              end
            INIT_LOAD:
              begin
              next = INIT_NOP4;
              state_cnt_nxt = 4'd1;
              end
            // INIT_NOP4: default - IDLE

            // REFRESH
            REF_PRE:
              begin
```

```
                    next = REF_NOP1;
                    end
                  REF_NOP1:
                    begin
                    next = REF_REF;
                    command_nxt = CMD_REF;
                    end
                  REF_REF:
                    begin
                    next = REF_NOP2;
                    state_cnt_nxt = 4'd7;
                    end
                  // REF_NOP2: default - IDLE

                  // WRITE
                  WRIT_ACT:
                    begin
                    next = WRIT_NOP1;
                    state_cnt_nxt = 4'd1;
                    end
                  WRIT_NOP1:
                    begin
                    next = WRIT_CAS;
                    command_nxt = CMD_WRIT;
                    end
                  WRIT_CAS:
                    begin
                    next = WRIT_NOP2;
                    state_cnt_nxt = 4'd1;
                    end
                  // WRIT_NOP2: default - IDLE

                  // READ
                  READ_ACT:
                    begin
                    next = READ_NOP1;
                    state_cnt_nxt = 4'd1;
                    end
                  READ_NOP1:
                    begin
                    next = READ_CAS;
                    command_nxt = CMD_READ;
                    end
                  READ_CAS:
                    begin
                    next = READ_NOP2;
                    state_cnt_nxt = 4'd1;
                    end
                  READ_NOP2:
                    begin
                    next = READ_READ;
                    end
                  // READ_READ: default - IDLE

                  default:
                    begin
                    next = IDLE;
                    end
                  endcase
              else
                begin
                // Counter Not Reached - HOLD
                next = state;
                command_nxt = command;
                end
        end

        endmodule

```