UNIVERSITY OF TEXAS AT ARLINGTON

Interfacing SDRAM with Intel® 80386DX™ (Verilog Source File)

Microprocessor Systems Design (Memory Interfacing)

Project Verilog Implementation on SDRAM controller for supporting MT48LC8M8A2 memory device and interfacing memory with an Intel® 80386DX™ Microprocessor.

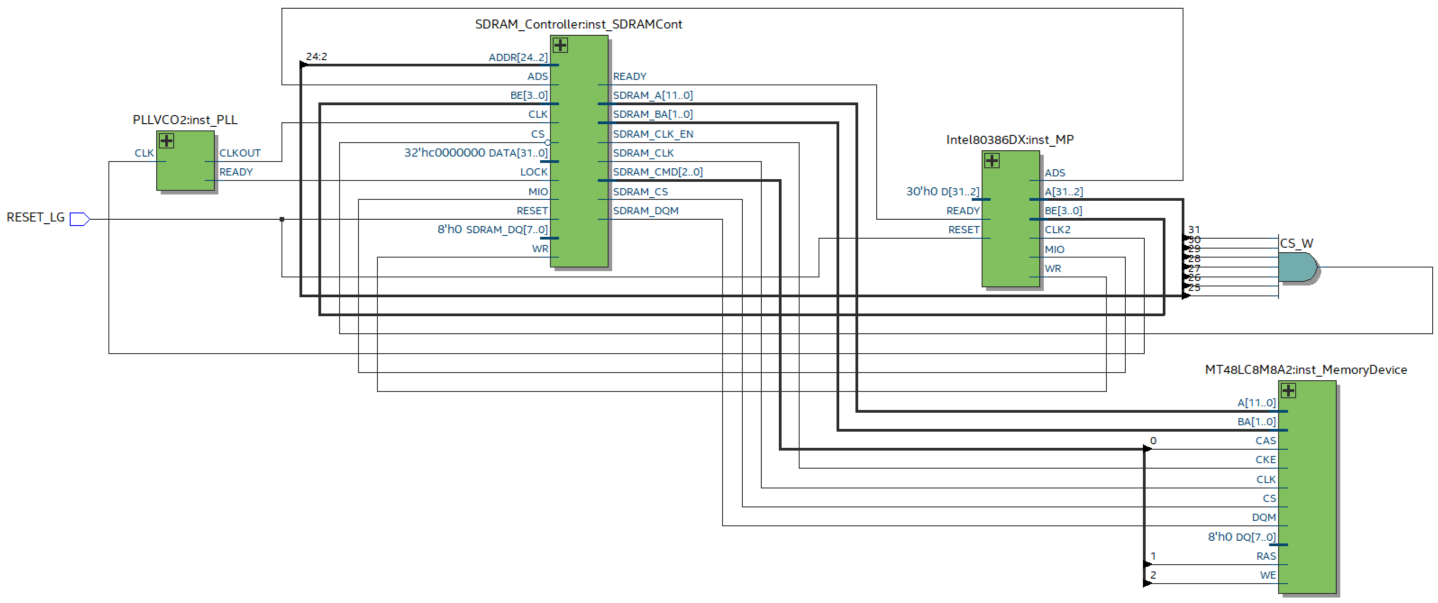
NABEEL NAYYAR • 9th August 2023 • UTA College of Engineering

SDRAM TOP-LEVEL DESIGN

TOP LEVEL DESIGN

```
module TopLevelDesign(
    input RESET_LG
);

    // Wiring CPU to SDRAM Controller
    wire CLKW;
    wire CLOCK_LOCK;
    wire CLKOUTW;
    wire [31:0] DATA_BUS;
    wire [31:2] ADDR_BUS;
    wire [3:0] BE_BUS;
    wire READY_W;
    wire WR_W;
    wire MIO_W;
    wire ADS_W;
    wire CS_W;

    wire RESET_W;
    assign RESET_W = RESET_LG;

    // Wiring SDRAM Controller to Memory Device
    wire SDRAM_CLKW;
    wire SDRAM_CKEW;
    wire [3:0] SDRAM_CMDW;
    wire [7:0] SDRAM_DATA_BUS;
    wire [11:0] SDRAM_ADDR_BUS;
    wire [1:0] SDRAM_BA_BUS;
    wire SDRAM_DQM;

    // Wiring for Data out Latch Controller
    // Clock Multiplication  PLL/VCO
    PLLVCO2 inst_PLL (
        .CLK    (CLKW),
        .READY  (CLOCK_LOCK),
        .CLKOUT (CLKOUTW)
    );
    // Intel 80386DX Block
    Intel80386DX inst_MP (
        .CLK2   (CLKW),              // CLK INPUT????
        .D      (DATA_BUS[31:0]),    // DATA 31-0
        .A      (ADDR_BUS),          // ADDR 31-2
        .BE     (BE_BUS),            // BE 3-0
        .WR     (WR_W),              // W/~R
        .DC     (),
        .MIO    (MIO_W),             // M/~IO
        .LOCK   (),
        .ADS    (ADS_W),             // ~ADS
        .NA     (),
        .READY  (READY_W),           // READY
        .BS16   (),
        .HOLD   (),
        .HLDA   (),
        .BUSY   (),
        .ERROR  (),
        .PEREQ  (),
        .INTR   (),
        .NMI    (),
        .RESET  (RESET_W)            // RESET
);

    // DCD Block
    assign CS_W = ~(ADDR_BUS[25] & ADDR_BUS[26] & ADDR_BUS[27] & ADDR_BUS[28] & ADDR_BUS[29]
& ADDR_BUS[30] & ADDR_BUS[31]);

    // SDRAM Controller Block
    SDRAM_Controller inst_SDRAMCont (
        .CLK    (CLKOUTW),
        .RESET  (RESET_W),          // [~RESET]
        .READY  (READY_W),          // READY to CPU
        .ADDR   (ADDR_BUS[24:2]),   // 30-bit Address Line from CPU
        .DATA   (DATA_BUS[31:0]),   // 32-bit Data Line from CPU
        .BE     (BE_BUS),           // [~BE] Bank Enable from CPU
        .ADS    (ADS_W),            // [~ADS] Address Strobe Signal from CPU
        .MIO    (MIO_W),            // [M/~IO]
        .WR     (WR_W),             // [W/~R]
        .CS     (CS_W),             // [~CS] >> FROM DCD BLOCK
        .LOCK   (CLOCK_LOCK),       // [~LOCK]
        // SDRAM -> Memory
        .SDRAM_CLK      (SDRAM_CLKW),   // To memory
        .SDRAM_CLK_EN   (SDRAM_CKEW),   // to memory
        .SDRAM_CMD      (SDRAM_CMDW[2:0]),  // SDRAM_CMD -> [~WE] [~RAS] [~CAS]
        .SDRAM_A        (SDRAM_ADDR_BUS),
        .SDRAM_DQ       (SDRAM_DATA_BUS[7:0]),
        .SDRAM_CS       (SDRAM_CMDW[3]),
        .SDRAM_BA       (SDRAM_BA_BUS),
        .SDRAM_DQM      (SDRAM_DQM)
    );

    // SDRAM Memory Block
    MT48LC8M8A2 inst_MemoryDevice (
        .CKE    (SDRAM_CKEW),
        .CLK    (SDRAM_CLKW),
        .CS     (SDRAM_CMDW[3]),
        .WE     (SDRAM_CMDW[2]),
        .CAS    (SDRAM_CMDW[0]),
        .RAS     (SDRAM_CMDW[1]),
        .DQM     (SDRAM_DQM),
        .BA      (SDRAM_BA_BUS),
        .A       (SDRAM_ADDR_BUS),
        .DQ      (SDRAM_DATA_BUS[7:0])
);

endmodule
```

RTL VIEW



SDRAM CONTROLLER

Clock Delay Module

```
// Shift Register clock cycle delay module

module delayed_Clock (
    input wire clk,
    input wire reset,
    input wire select,
    input wire input_signal,
    output wire output_logic
);

reg [6:0] shift_registerA = 0;  // 0
reg [1:0] shift_registerB = 0;  // 1
wire delayed_signal;

initial begin
    if (!select) begin
        assign delayed_signal = shift_registerA[6];
    end
    else begin
        assign delayed_signal = shift_registerB[1];
    end
end

always_ff @ (posedge clk or posedge reset) begin
    // 7 Cycles delay
     if (reset && !select)
        shift_registerA <= 0;
    else
        shift_registerA <= {shift_registerA[5:0], input_signal};

     // 2 cycles delay
     if (reset && select)
        shift_resisterB <= 0;
     else
        shift_registerB <= {shift_registerB[0], input_signal};
end

assign output_logic = delayed_signal;

endmodule
```

Multiplexer Module

```
// Generic 4-1 MUX Module

module MUX (
    input wire [3:0] A,
    input wire [1:0] S,
    output wire OUT
);

assign mux_output =     (S == 2'b00) ? A[0] :
                        (S == 2'b01) ? A[1] :
                        (S == 2'b10) ? A[2] :
                       A[3];

endmodule
```

Generic Latch Module

```
// Simple Latch Module for DATA Bus

module S_LATCH (
    input [8:0]IN,      // Data input
    input G,                // Set input (active high)
    input OE,           // Output enable input (active low)
    output [8:0]OUT // Data Output
);

    always @(IN, G, OE)
    begin
        if (!OE)
        begin
            if (G)
                OUT <= IN; // Set the output to the input value when G is asserted
            // Note: If G is not asserted (G=0), the output retains its previous value.
        end
    end


endmodule
```

Generic Count-down counter module

```
// Generic Count down timer module

module WAIT_ST_COUNTER (
        input CLK,          // Clock INPUT
        input LOAD,         // ~RESET state to Load Value
        input [15:0] X,     // Input Load Value
        output ZERO     // Zero Counter
);


    reg [31:0] countREG; // Internal register to count down X

    always_ff @(posedge CLK or negedge LOAD) begin
        if (!LOAD) // Reset the counter to load value
            countREG <= X;
        else if (countREG > 0)
            countREG <= countREG - 1;
    end

    assign ZERO = (countREG == 0); // Assign 1 iff 0
    // Be mindful of the indexing
endmodule
```

SDRAM Controller Diagram

```
// (C) Nabeel Nayyar | The University of Texas
// Microprocessor Systems Design
//
// Top Level Design of the SDRAM Controller

module SDRAM_Controller(
            // CPU -> 80386DX
            input CLK,
            input RESET,                    // [~RESET]
            output READY,
            input [24:2] ADDR,              // 30-bit Address Line from CPU
            inout [31:0] DATA,              // 32-bit Data Line from CPU
            input [3:0] BE,                 // [~BE] Bank Enable from CPU
            input ADS,                      // [~ADS] Address Strobe Signal from CPU
            input MIO,                      // [M/~IO]
            input WR,                       // [W/~R]
            input CS,                       // [~CS]
            input LOCK,                     // [~LOCK] PLL Lock

            // SDRAM -> Memory
            output SDRAM_CLK,
            output SDRAM_CLK_EN,
            output [2:0] SDRAM_CMD, // SDRAM_CMD -> [~RAS] [~CAS] [~WE]
            output [11:0] SDRAM_A,
            inout [7:0] SDRAM_DQ,
            output SDRAM_CS,
            output [1:0] SDRAM_BA,
            output SDRAM_DQM
);
    // ---------- Generating MWTC & MRDC Signals ----------
    wire MWTC_n;    // Memory Write
    wire MRDC_n;    // Memort Read
    always_ff @ (posedge CLK) begin
        if (CLK) begin
            assign MWTC_n = ~(MIO & WR & ADS);  // MWTC Always generated
            assign MRDC_n = ~(MIO & ~WR & ~ADS);
        end
    end

    // Connecting input CLK to SDRAM_CLK (100 MHz) [ASSUMPTION]
    assign SDRAM_CLK = CLK;

    // TOP-Level STATES
    parameter INIT      = 3'd0;
    parameter IDLE      = 3'd1;
    parameter REFRESH   = 3'd2;
    parameter WRITE     = 3'd3;
    parameter READ          = 3'd4;
    parameter IDLE_S        = 3'd5;

    reg [8:0] M_STATE;  // Holder of Main state

    // Low-Level STATES (INITIALIZATION)    tRP = 2cyc | tRFC = 7cyc | tMRD = 2cyc
    parameter INIT_RESET = 6'd6;
    parameter INIT_WAIT     = 6'd7;
    parameter INIT_PC       = 6'd8;
    parameter INIT_NOP0 = 6'd9;
    parameter INIT_ARF0 = 6'd10;
    parameter INIT_NOP1 = 6'd11;
    parameter INIT_ARF1 = 6'd12;
    parameter INIT_NOP2 = 6'd13;
    parameter INIT_LMR  = 6'd14;
    parameter INIT_NOP3 = 6'd15;
    // Low-Level STATES (REFRESH)
    parameter REF_ARF0  = 6'd16;
    parameter REF_NOP0  = 6'd17;
    parameter REF_NOP1  = 6'd18;
    // Low-Level STATES (READ)
    parameter RD_ACT        = 6'd19;
    parameter RD_NOP0       = 6'd20;
    parameter RD_RDCM       = 6'd37;
    parameter RD_NOP1       = 6'd38;
    parameter RD_RD1        = 6'd21;
    parameter RD_NOP2       = 6'd22;
    parameter RD_RD2        = 6'd23;
    parameter RD_RD3        = 6'd24;
    parameter RD_RD4        = 6'd25;
    parameter RD_NOP3       = 6'd26;
    parameter RD_PC     = 6'd27;
    // Low-Lever STATES (WRITE)
    parameter WR_ACT        = 6'd28;
    parameter WR_NOP0       = 6'd29;
    parameter WR_WR1        = 6'd30;
    parameter WR_WR2        = 6'd31;
    parameter WR_WR3        = 6'd32;
    parameter WR_WR4        = 6'd33;
    parameter WR_NOP1       = 6'd34;
    parameter WR_PC     = 6'd35;
    parameter WR_NOP2       = 6'd36;

    // SDRAM Command Parameters (RAS#, CAS#, WE#)
    parameter CMD_NOP   = 3'b111;
    parameter CMD_ACT       = 3'b011;
    parameter CMD_RD        = 3'b101;
    parameter CMD_WR        = 3'b100;
    parameter CMD_PC        = 3'b010;
    parameter CMD_ARF       = 3'b001;
    parameter CMD_LMR       = 3'b000;
    // LMR OPCODE for initializing the SDRAM memory
    parameter LMR_OPCODE    = 10'b000000100010;

    // Local State Identifier
    reg [64:0] L_STATE;

    // Inialization Parameters and wiring
    parameter X = 16'b0010011100010000; // Load Value for TIMER0 (100us) = 10k Cyc
    parameter Y = 16'b0000011000011000; // Load Value for TIMER1 = 1560 Cyc
    wire RD_1;  // Signal wire on completion of RD_1 L_STATE
    wire RD_2;  // Signal wire on completion of RD_2 L_STATE
    wire RD_3;  // Signal wire on completion of RD_3 L_STATE
    wire RD_4;  // Signal wire on completion of RD_4 L_STATE

    // ---------- TIMER 0 (100us Delay) Logic ----------
    wire TMR_RESET_W;
    wire TIMEOUT;
    WAIT_ST_COUNTER inst_TIMER0 (       // Count till 100us -> 10,000 cyc
        .CLK    (CLK),
        .LOAD (TMR_RESET_W),
        .X  (X),
        .ZERO (TIMEOUT)
    );

    // ---------- TIMER 1 (Interrupt for refresh) Logic ----------
```

```
wire TIMEOUT1;
wire LOAD_WT1;
wire REFRESH_REQUEST;   // External Parameter of REFRESH_REQUESTED
wire CLR_R_R;               // Clear Refresh Request
WAIT_ST_COUNTER inst_TIMER1 (       // Count till 10,000 cyc
    .CLK    (CLK),
    .LOAD (LOAD_WT1),
    .X  (Y),
    .ZERO (TIMEOUT1)
);
always_ff @ (posedge CLK) begin
    assign LOAD_WT1 = RESET || TIMEOUT1;
    if (TIMEOUT1) begin // SET logic
        assign REFRESH_REQUEST = 1'b1;
    end
    else if (CLR_R_R) begin
    // Clear request and restart timer
        assign REFRESH_REQUEST = 1'b0;
        assign LOAD_WT1 = 1'b0;
    end
end

// ---------- Clock delay logic ----------
wire DELAY_REST;
wire DELAY_SEL;
wire DELAY_OUT;
delayed_Clock inst_wait(
    .clk(CLK),
    .reset(DELAY_REST),
    .select(DELAY_SEL), // 0 = 7cyc loss | 1 = 2cyc loss
    .input_signal(1'b1),
    .output_logic(DELAY_OUT)
);

// ---------- DQM State logic ----------
wire DQM_MUX_SEL;
wire DQM_OUT_W;
MUX inst_DQMSL (
    .A      (BE),
    .S      (DQM_MUX_SEL),
    .OUT    (DQM_OUT_W)
);
always_ff @ (posedge CLK) begin

    // DQM State function on [WRITE STATE]
    if (M_STATE == WRITE && L_STATE == WR_WR1 && !BE[0]) begin
        assign DQM_MUX_SEL = 2'b00; // DQM 0, Valid Write on D[7:0]
    end
    else if (M_STATE == WRITE && L_STATE == WR_WR2 && !BE[1]) begin
        assign DQM_MUX_SEL = 2'b01; // DQM 0, Valid Write on D[15:8]
    end
    else if (M_STATE == WRITE && L_STATE == WR_WR3 && !BE[2]) begin
        assign DQM_MUX_SEL = 2'b10; // DQM 0, Valid Write on D[23:16]
    end
    else if (M_STATE == WRITE && L_STATE == WR_WR4 && !BE[3]) begin
        assign DQM_MUX_SEL = 2'b11;
    end
    else begin
        assign DQM_OUT_W = 1'b1;    // Bypass Logic to assert 1
    end
    // DQM State function on [READ STATE]
    if (M_STATE == READ && L_STATE == RD_RD1 && !BE[0]) begin
        assign DQM_MUX_SEL = 2'b00; // DQM 0, Valid Read on D[7:0]
    end
    else if (M_STATE == READ && L_STATE == RD_RD2 && !BE[1]) begin
        assign DQM_MUX_SEL = 2'b01; // DQM 0, Valid Read on D[15:8]
    end
    else if (M_STATE == READ && L_STATE == RD_RD3 && !BE[2]) begin
        assign DQM_MUX_SEL = 2'b10; // DQM 0, Valid Read on D[23:16]
    end
    else if (M_STATE == READ && L_STATE == RD_RD4 && !BE[3]) begin
        assign DQM_MUX_SEL = 2'b11;
    end
    else begin
        assign DQM_OUT_W = 1'b1;    // Bypass logic to assert 1
    end
end

// ****** Main Top-Level FSM ******
always_ff @ (posedge CLK) begin
    // Boot up to INIT_RESET STATE
    if (!RESET) begin           // Boot up
        SDRAM_CLK_EN = 1'b0;        // Assert CLK_EN to stay off
        L_STATE <= INIT_RESET;  // Assign RESET to Local STATE
        M_STATE <= INIT;            // Assign INIT to Main STATE
        READY = 1;                  // Not Ready
    end
    else begin
        case (M_STATE)
            // IDLE State
            IDLE:
                begin
                    READY = 0;              // Ready for CPU Instruction
                    SDRAM_CLK_EN = 1'b1;    // Re-assert CLK_EN to stay on
                    SDRAM_CMD = CMD_NOP;    // Issue NOP to the SDRAM
                    SDRAM_CS = 1'b0;
                    if (!MWTC_n) begin  // Memory Write Command
                        M_STATE <= WRITE;
                    end
                    else if (!MRDC_n) begin // Memory Read Command
```

```
                            M_STATE <= READ;
                        end
                        else if (REFRESH_REQUEST) begin // Check for Refresh request
                            M_STATE <= REFRESH;
                        end
                        else begin
                            M_STATE <= IDLE;    // Else remain in IDLE state
                        end
                    end
                // Refresh State
                REFRESH:
                    begin
                        SDRAM_CLK_EN = 1'b1;    // Re-assert CLK_EN to stay on
                        READY = 1'b1;           // Not Ready for CPU instruction
                        L_STATE <= REF_ARF0;    // Change Local State to Refresh
                    end
                // Read State
                READ:
                    begin
                        SDRAM_CLK_EN = 1'b1;    // Re-assert CLK_EN to stay on
                        READY = 1'b1;           // Complying with CPU instruction, !READY
                        L_STATE <= RD_ACT;  // Change Local State to READ
                    end
                // Write State
                WRITE:
                    begin
                        SDRAM_CLK_EN = 1'b1;    // Re-assert CLK_EN to stay on
                        READY = 1'b1;           // Complying with CPU instruction, !READY
                        L_STATE <= WR_ACT;
                    end
                // Ghost IDLE state for READY
                S_IDLE:
                    begin
                        READY = 1'b0;           // Ready for CPU Instruction
                        SDRAM_CLK_EN = 1'b1;    // Re-assert CLK_EN to stay on
                        SDRAM_CMD = CMD_NOP;    // Issue NOP to the SDRAM
                        SDRAM_CS = 1'b0;
                        if (!MWTC_n) begin  // Memory Write Command
                            M_STATE <= WRITE;
                        end
                        else if (!MRDC_n) begin // Memory Read Command
                            M_STATE <= READ;
                        end
                        else if (REFRESH_REQUEST) begin // Check for Refresh request
                            M_STATE <= REFRESH;
                        end
                        else begin
                            READY = 1'b0;               // Just to be sure yk
                            M_STATE <= IDLE;        // Return to IDLE state
                        end
                    end
                endcase
        end
    end


    // Initialization FSM
    always_comb @ (*) begin
        if (M_STATE == INIT && L_STATE == INIT_RESET) begin
            if (LOCK) begin
                L_STATE = INIT_WAIT;
                assign TMR_RESET_W = 1'b0;      // Start Timer

                // Start the case for the FSM
                begin
                    case (L_STATE)
                        // Initial WAIT State
                        INIT_WAIT:                          // Wait till the TIMEOUT
                            begin
                                SDRAM_CLK_EN = 1'b1;        // Enable SDRAM Clock
                                SDRAM_DQM = 1'bx;           // Asserting DQM as x
                                if (TIMEOUT) begin      // Wait till TIMEOUT
                                    L_STATE <= INIT_PC; // Move to the Next State
                                end
                            end
                        // Precharge All Banks State
                        INIT_PC:
                            begin
                                SDRAM_CMD = CMD_PC;     // Issue a Precharge command
                                SDRAM_CS = 1'b0;
                                SDRAM_A[10] = 1'b1;
                                SDRAM_DQM = 1'bx;
                                L_STATE <= INIT_NOP0;   // Move to the Next State
                            end
                        // NOP State till tRP = 2
                        INIT_NOP0:
                            begin
                                SDRAM_CMD = CMD_NOP;        // issue an Auto Refresh Command
                                SDRAM_CS = 1'b0;
                                assign DELAY_SEL = 1'b1;
                                assign DELAY_REST = 1'b1;
                                // Anticipate 2 cycle loss till assignment complete
                                if (DELAY_OUT) begin
                                    L_STATE <= INIT_ARF0;
                                end
                            end
                        // Auto Refresh State
                        INIT_ARF0:
                            begin
                                SDRAM_CMD = CMD_ARF;   // Issue an Auto Refresh Command
                                SDRAM_CS = 1'b0;
                                SDRAM_A[10] = 1'bx;
                                SDRAM_DQM = 1'bx;
                                L_STATE <= INIT_NOP1;
                            end
                        // NOP State till tRFC
                        INIT_NOP1:
                            begin
                                SDRAM_CMD = CMD_NOP;            // Issue a NOP command
                                SDRAM_CS = 1'b0;
                                assign DELAY_SEL = 1'b0;
                                assign DELAY_REST = 1'b1;
                                // Anticipate 7 cycle loss till assignment complete
                                if (DELAY_OUT) begin
                                    L_STATE <= INIT_ARF1;
                                end
                            end
                        // Auto Refresh State
                        INIT_ARF1:
                            begin
                                SDRAM_CMD = CMD_ARF;        // Issue an Auto Refresh Command
                                SDRAM_CS = 1'b0;
                                SDRAM_A[10] = 1'bx;
                                SDRAM_DQM = 1'bx;
                                L_STATE <= INIT_NOP2;   // Move to the Next state
                            end
                        // NOP State till tRFC
                        INIT_NOP2:
                            begin
                                SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                                SDRAM_CS = 1'b0;
                                assign DELAY_SEL = 1'b0;
                                assign DELAY_REST = 1'b1;
                                // Anticipate 7 cycle loss till assignment complete
                                if (DELAY_OUT) begin
                                    L_STATE <= INIT_LMR;    // Move to the next state
                                end
                            end
                        // LMR State
                        INIT_LMR:
                            begin
                                SDRAM_CMD = CMD_LMR;        // Issue an LMR command
                                SDRAM_CS    = 1'b0;
                                SDRAM_A <= LMR_OPCODE;  // Load the OPCODE to SDRAM_A
                                SDRAM_DQM = 1'bx;
                                L_STATE <= INIT_NOP3;   // Move to the Next State
                            end
                        // NOP till tMRD
                        INIT_NOP3:
                            begin
                                SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                                SDRAM_CS = 1'b0;
                                assign DELAY_SEL = 1'b1;
                                assign DELAY_REST = 1'b1;
                                // Anticipate 2 cycle loss till assignment complete
                                if (DELAY_OUT) begin
                                    M_STATE <= IDLE;        // Entered IDLE state
                                    L_STATE <= 1'bx;        // Dont care now
                                end
                            end
                    endcase
                end
            end
        end
    end



    // Refresh FSM
    always_comb @ (*) begin
        if (M_STATE == REFRESH && L_STATE == REF_ARF0) begin
            // Initiate an AUTO REFRESH Command
                SDRAM_CMD = CMD_ARF;        // Issue an AUTO REFRESH command
                SDRAM_CS = 1'b0;
                SDRAM_A[10] = 1'bx;
                SDRAM_DQM = 1'bx;
                L_STATE <= REF_NOP0;        // Move to the next state
        end
        case (L_STATE)
            // Singular NOP to SDRAM
            REF_NOP0:
                begin
                    SDRAM_CMD = CMD_NOP;    // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= REF_NOP1;    // Move to the Next state
                end
            REF_NOP1:
                begin
                    SDRAM_CMD = CMD_NOP;    // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= 1'bx;        // Local State dont care
                    CLR_R_R = 1'b0;     // Clear the Refresh Request
                    M_STATE <= IDLE;
                end
        endcase
    end

    // Read FSM | timing constrains CL = 2
    always_comb @ (*) begin
        if (M_STATE == READ && L_STATE == RD_ACT) begin
```

```
            // Activate Command
            READY = 1'b1;                   // Re-assert !READY
            SDRAM_CMD = CMD_ACT;            // Issue a ACTIVE command
            SDRAM_CS = 1'b0;
            SDRAM_A = ADDR[22:11];      // Selecting ROW Address from CPU ADDR
            SDRAM_BA[0] = ADDR[23];     // ADDR [23] -> BA 0 (Selecting Banks)
            SDRAM_BA[1] = ADDR[24];     // ADDR [24] -> BA 1
            SDRAM_DQM = 1'bx;               // DQM at the moment dont care
            L_STATE <= RD_NOP0;
        end
        case (L_STATE)
            // NOP Command
            RD_NOP0:
                begin
                    SDRAM_CMD = CMD_NOP;    // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= RD_RD;       // Move to the next state
                end
            // READ Command
            RD_RDCM:
                begin
                    SDRAM_CMD = CMD_RD;         // Issue a READ command
                    SDRAM_CS = 1'b0;
                    SDRAM_A = ADDR[10:2];       // Selecting COL Address from CPU ADDR (A0-
A9)
                    SDRAM_A[10] = 1'b0;             // Disable Auto Precharge
                    // Assigning Bank Adresses to the SDRAM BA pins
                    SDRAM_BA[0] = ADDR[23];     // ADDR [23] -> BA 0
                    SDRAM_BA[1] = ADDR[24];     // ADDR [24] -> BA 1
                    assign SDRAM_DQM = DQM_OUT_W;   // DQM the function of ~BE3-0
                    assign RD_1 = 1'b1;         // Activate Gate Latch for D7-0
                    L_STATE <= RD_NOP1;
                end
            // NOP till CL (2 cyc loss)
            RD_NOP1:
                begin
                    SDRAM_CMD = CMD_NOP;        // issue a NOP Command
                    SDRAM_CS = 1'b0;
                    assign DELAY_SEL = 1'b1;
                    assign DELAY_REST = 1'b1;
                    // Anticipate 2 cycle loss till assignment complete
                    if (DELAY_OUT) begin
                        L_STATE <= RD_RD1;
                    end
                end
            // Start Reading SDRAM Bursts
            RD_RD1:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP Command
                    SDRAM_CS = 1'b0;
                    assign RD_1 = 1'b1;     // Gate Latch for Reading D7-0
                    L_STATE <= RD_RD2;
                end
            RD_RD2:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP Command
                    SDRAM_CS = 1'b0;
                    assign RD_2 = 1'b1;     // Gate Latch for Reading D15-8
                    L_STATE <= RD_RD3;
                end
            RD_RD3:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP Command
                    SDRAM_CS = 1'b0;
                    assign RD_3 = 1'b1;     // Gate Latch for Reading D23-16
                    L_STATE <= RD_RD4;
                end
            RD_RD4:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP Command
                    SDRAM_CS = 1'b0;
                    assign RD_4 = 1'b1;     // Gate Latch for Reading D31-24
                    L_STATE <= RD_NOP2;
                end
            // READY is asserted and NOP state [DATA IS READY]
            RD_NOP2:
                begin
                    SDRAM_CMD = CMD_NOP;        // issue a NOP Command
                    SDRAM_CS = 1'b0;
                    L_STATE <= RD_PC;
                    READY = 1'b0;               // We start Asserting READY to meet CPU
Timing
                end
            // Precharge All banks after READ
            RD_PC:
                begin
                    SDRAM_CMD = CMD_PC;     // Issue a Precharge command
                    SDRAM_CS = 1'b0;
                    SDRAM_A[10] = 1'b1;
                    SDRAM_DQM = 1'bx;
                    L_STATE <= RD_NOP3;     // Move to the Next State
                    READY = 1'b0;               // We start Asserting READY to meet CPU
Timing
                end
            // NOP to Ghost IDLE state
            RD_NOP3:
                begin
                    SDRAM_CMD = CMD_NOP;        // issue a NOP Command
                    SDRAM_CS = 1'b0;
                    L_STATE <= 1'dx;
                    READY = 1'b0;               // We start Asserting READY to meet CPU
Timing
                    M_STATE <= S_IDLE;
                end
        endcase
    end

    // Write FSM
    always_comb @ (*) begin
        // Issue an Active Command
        if (M_STATE == WRITE && LSTATE == WR_ACT) begin
            // Activate Command
            READY = 1'b1;                   // Re-assert !READY
            SDRAM_CMD = CMD_ACT;            // Issue a ACTIVE command
            SDRAM_CS = 1'b0;
            SDRAM_A = ADDR[22:11];      // Selecting ROW Address from CPU ADDR BUS
            SDRAM_BA[0] = ADDR[23];     // ADDR [23] -> BA 0
            SDRAM_BA[1] = ADDR[24];     // ADDR [24] -> BA 1
            SDRAM_DQM = 1'bx;               // DQM is dont care
            L_STATE <= WR_NOP0;
        end
        case (L_STATE)
            // NOP to Write first burst
            WR_NOP0:
                begin
                    SDRAM_CMD = CMD_NOP;            // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= WR_WR1;
                end
            // Write in Bursts (Little endian -> Intel M.P support)
            WR_WR1:     // Burst 1
                begin
                    SDRAM_CMD = CMD_WR;         // Issue a Write Command
                    SDRAM_CS = 1'b0;
                    SDRAM_DQM = DQM_OUT_W;      // Assign DQM by the function of STATE (LOW)
to Write
                    SDRAM_A[8:0] = ADDR[10:2];  // Selecting COL Address from CPU ADDR BUS
                    SDRAM_A[9]   = 1'bx;
                    SDRAM_A[10] = 1'b1;         // A10 = 1 for Auto Precharge
                    SDRAM_A[11] = 1'bx;
                    // Assigning Bank Address to the BA of SDRAM
                    SDRAM_BA[0] = ADDR[23];     // ADDR [23] -> BA 0
                    SDRAM_BA[1] = ADDR[24];     // ADDR [24] -> BA 1
                    // Write DATA 0-7 to Memory Data Bus
                    assign SDRAM_D[7:0] = DATA[0:7];
                    L_STATE <= WR_WR2;
                end
            WR_WR2:     // Burst 2
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    // Sequential DATA 8-15 write (NO CMD REQUIRED)
                    assign SDRAM_D[7:0] = DATA[8:15];
                    L_STATE <= WR_WR3;      // Move to the next state
                end
            WR_WR3:     // Burst 3
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    // Sequential DATA 16-23 write (NO CMD REQUIRED)
                    assign SDRAM_D[7:0] = DATA[16:23];
                    L_STATE <= WR_WR4;      // Move to the next state
                end
            WR_WR4:     // Burst 4
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    // Sequential DATA 24-31 write (NO CMD REQUIRED)
                    assign SDRAM_D[7:0] = DATA[24:31];
                    L_STATE <= WR_NOP1;     // Move to the next state
                end
            WR_NOP1:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= WR_PC;           // Move to the next state
                end
            WR_PC:
                begin
                    SDRAM_CMD = CMD_PC;     // Issue a Precharge command
                    SDRAM_CS = 1'b0;
                    SDRAM_A[10] = 1'b1;
                    SDRAM_DQM = 1'bx;
                    READY = 1'b0;           // We start Asserting READY to meet CPU Timing
                    L_STATE <= WR_NOP2;     // Move to the Next State
                end
            WR_NOP2:
                begin
                    SDRAM_CMD = CMD_NOP;        // Issue a NOP command
                    SDRAM_CS = 1'b0;
                    L_STATE <= 1'dx;            // Dont care Local State
                    READY = 1'b0;            // We start Asserting READY to meet CPU Timing
                    M_STATE <= S_IDLE;      // Main State Change to Ghost IDLE
                end
        endcase
    end

    // High-Z on DATA if we are in a state w no DATA anticipation
    always_ff @ (posedge CLK) begin
        if (M_STATE == INIT || M_STATE == REFRESH) begin
```

```
            // Shut off CPU Data Bus
            assign DATA = 32'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz;
            // Shut off SDRAM Data bus
            assign SDRAM_DQ = 8'bzzzzzzzz;
        end
    end

    // -------------- Data OUT Latch Logic --------------
    // Latch for D7-0
    S_LATCH inst_D7to0 (
        .IN (SDRAM_DQ),     // Data input
        .G      (RD_1),         // Set input (active high)
        .OE (~READY),       // Output enable input (active low)
        .OUT    (DATA[7:0])     // Data Output
    );

    // Latch for D15-8
    S_LATCH inst_D15to8 (
        .IN (SDRAM_DQ),     // Data input
        .G      (RD_2),         // Set input (active high)
        .OE (~READY),       // Output enable input (active low)
        .OUT    (DATA[15:8])    // Data Output
    );

        // Latch for D23-16
    S_LATCH inst_D23to16 (
        .IN (SDRAM_DQ),     // Data input
        .G      (RD_3),         // Set input (active high)
        .OE (~READY),       // Output enable input (active low)
        .OUT    (DATA[23:16])   // Data Output
    );

        // Latch for D31-24
    S_LATCH inst_D31to24 (
        .IN (SDRAM_DQ),     // Data input
        .G      (RD_4),         // Set input (active high)
        .OE (~READY),       // Output enable input (active low)
        .OUT    (DATA[31:24])   // Data Output
    );

endmodule
```