

```

// *****
// **
// ** M24LC64.v - 24LC64 64K-BIT I2C SERIAL EEPROM (VCC = +2.5V TO +5.5V)
// **
// *****
// **
// **          COPYRIGHT (c) 2003 YOUNG ENGINEERING
// **          ALL RIGHTS RESERVED
// **
// ** THIS PROGRAM IS CONFIDENTIAL AND A TRADE SECRET OF YOUNG ENGINEERING. THE RECEIPT OR
// ** POSSESSION OF THIS PROGRAM DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS
// ** CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE, IN WHOLE OR IN
// ** PART, WITHOUT THE SPECIFIC WRITTEN CONSENT OF YOUNG ENGINEERING.
// **
// *****
// ** Revision      : 1.1
// ** Modified Date : 07/19/2004
// ** Revision History:
// **
// ** 10/01/2003:  Initial design
// ** 07/19/2004:  Fixed the timing checks and the open-drain modeling for SDA.
// **
// *****
// **          TABLE OF CONTENTS
// **
// ** -----
// ** DECLARATIONS
// ** -----
// **
// ** INITIALIZATION
// ** -----
// **
// ** CORE LOGIC
// ** -----
// **
// ** 1.01:  START Bit Detection
// ** 1.02:  STOP Bit Detection
// ** 1.03:  Input Shift Register
// ** 1.04:  Input Bit Counter
// ** 1.05:  Control Byte Register
// ** 1.06:  Byte Address Register
// ** 1.07:  Write Data Buffer
// ** 1.08:  Acknowledge Generator
// ** 1.09:  Acknowledge Detect
// ** 1.10:  Write Cycle Timer
// ** 1.11:  Write Cycle Processor
// ** 1.12:  Read Data Multiplexor
// ** 1.13:  Read Data Processor
// ** 1.14:  SDA Data I/O Buffer
// **
// ** -----
// ** DEBUG LOGIC
// ** -----
// **
// ** 2.01:  Memory Data Bytes
// ** 2.02:  Write Data Buffer
// **
// ** -----
// ** TIMING CHECKS
// ** -----
// **
// *****

```

`timescale 1ns/10ps

```

module M24LC64 (A0, A1, A2, WP, SDA, SCL, RESET);

```

```

    input          A0;           // chip select bit
    input          A1;           // chip select bit
    input          A2;           // chip select bit

    input          WP;           // write protect pin

    inout          SDA;          // serial data I/O
    input          SCL;          // serial data clock

    input          RESET;        // system reset

```

```

// *****
// ** DECLARATIONS
// *****

```

```

reg          SDA_DO;                // serial data - output
reg          SDA_OE;                // serial data - output enable

wire         SDA_DriveEnable;       // serial data output enable
reg          SDA_DriveEnableDlyd;   // serial data output enable - delayed

wire [02:00] ChipAddress;           // hardwired chip address

reg [03:00]  BitCounter;            // serial bit counter

reg          START_Rcvd;            // START bit received flag
reg          STOP_Rcvd;             // STOP bit received flag
reg          CTRL_Rcvd;             // control byte received flag
reg          ADHI_Rcvd;             // byte address hi received flag
reg          ADLO_Rcvd;             // byte address lo received flag
reg          MACK_Rcvd;             // master acknowledge received flag

reg          WrCycle;               // memory write cycle
reg          RdCycle;               // memory read cycle

reg [07:00]  ShiftRegister;        // input data shift register

reg [07:00]  ControlByte;          // control byte register
wire         RdWrBit;              // read/write control bit

reg [12:00]  StartAddress;         // memory access starting address
reg [04:00]  PageAddress;         // memory page address

reg [07:00]  WrDataByte [0:31];    // memory write data buffer
wire [07:00] RdDataByte;          // memory read data

reg [12:00]  WrCounter;            // write buffer counter

reg [04:00]  WrPointer;            // write buffer pointer
reg [12:00]  RdPointer;            // read address pointer

reg          WriteActive;          // memory write cycle active

reg [07:00]  MemoryBlock0 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock1 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock2 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock3 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock4 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock5 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock6 [0:1023]; // EEPROM data memory array
reg [07:00]  MemoryBlock7 [0:1023]; // EEPROM data memory array

integer      LoopIndex;            // iterative loop index

integer      tAA;                  // timing parameter
integer      tWC;                  // timing parameter

// *****
// **   INITIALIZATION   **
// *****

initial tAA = 900;                  // SCL to SDA output delay
initial tWC = 5000000;             // memory write cycle time

initial begin
    SDA_DO = 0;
    SDA_OE = 0;
end

initial begin
    START_Rcvd = 0;
    STOP_Rcvd = 0;
    CTRL_Rcvd = 0;
    ADHI_Rcvd = 0;
    ADLO_Rcvd = 0;
    MACK_Rcvd = 0;
end

initial begin
    BitCounter = 0;
    ControlByte = 0;
end

initial begin
    WrCycle = 0;
    RdCycle = 0;

```

```

    WriteActive = 0;
end

assign ChipAddress = {A2,A1,A0};

// *****
// ** CORE LOGIC **
// *****
// -----
// 1.01: START Bit Detection
// -----

always @(negedge SDA) begin
    if (SCL == 1) begin
        START_Rcvd <= 1;
        STOP_Rcvd <= 0;
        CTRL_Rcvd <= 0;
        ADHI_Rcvd <= 0;
        ADLO_Rcvd <= 0;
        MACK_Rcvd <= 0;

        WrCycle <= #1 0;
        RdCycle <= #1 0;

        BitCounter <= 0;
    end
end

// -----
// 1.02: STOP Bit Detection
// -----

always @(posedge SDA) begin
    if (SCL == 1) begin
        START_Rcvd <= 0;
        STOP_Rcvd <= 1;
        CTRL_Rcvd <= 0;
        ADHI_Rcvd <= 0;
        ADLO_Rcvd <= 0;
        MACK_Rcvd <= 0;

        WrCycle <= #1 0;
        RdCycle <= #1 0;

        BitCounter <= 10;
    end
end

// -----
// 1.03: Input Shift Register
// -----

always @(posedge SCL) begin
    ShiftRegister[00] <= SDA;
    ShiftRegister[01] <= ShiftRegister[00];
    ShiftRegister[02] <= ShiftRegister[01];
    ShiftRegister[03] <= ShiftRegister[02];
    ShiftRegister[04] <= ShiftRegister[03];
    ShiftRegister[05] <= ShiftRegister[04];
    ShiftRegister[06] <= ShiftRegister[05];
    ShiftRegister[07] <= ShiftRegister[06];
end

// -----
// 1.04: Input Bit Counter
// -----

always @(posedge SCL) begin
    if (BitCounter < 10) BitCounter <= BitCounter + 1;
end

// -----
// 1.05: Control Byte Register
// -----

always @(negedge SCL) begin
    if (START_Rcvd & (BitCounter == 8)) begin
        if (!WriteActive & (ShiftRegister[07:01] == {4'b1010,ChipAddress[02:00]})) begin
            if (ShiftRegister[00] == 0) WrCycle <= 1;
            if (ShiftRegister[00] == 1) RdCycle <= 1;
        end
    end
end

```

```

        ControlByte <= ShiftRegister[07:00];

        CTRL_Rcvd <= 1;
    end

    START_Rcvd <= 0;
end
end

assign RdWrBit = ControlByte[00];

// -----
//      1.06:  Byte Address Register
// -----

always @(negedge SCL) begin
    if (CTRL_Rcvd & (BitCounter == 8)) begin
        if (RdWrBit == 0) begin
            StartAddress[12:08] <= ShiftRegister[04:00];
            RdPointer[12:08]    <= ShiftRegister[04:00];

            ADHI_Rcvd <= 1;
        end

        WrCounter <= 0;
        WrPointer <= 0;

        CTRL_Rcvd <= 0;
    end
end

always @(negedge SCL) begin
    if (ADHI_Rcvd & (BitCounter == 8)) begin
        if (RdWrBit == 0) begin
            StartAddress[07:00] <= ShiftRegister[07:00];
            RdPointer[07:00]    <= ShiftRegister[07:00];

            ADLO_Rcvd <= 1;
        end

        WrCounter <= 0;
        WrPointer <= 0;

        ADHI_Rcvd <= 0;
    end
end

// -----
//      1.07:  Write Data Buffer
// -----

always @(negedge SCL) begin
    if (ADLO_Rcvd & (BitCounter == 8)) begin
        if (RdWrBit == 0) begin
            WrDataByte[WrPointer] <= ShiftRegister[07:00];

            WrCounter <= WrCounter + 1;
            WrPointer <= WrPointer + 1;
        end
    end
end

// -----
//      1.08:  Acknowledge Generator
// -----

always @(negedge SCL) begin
    if (!WriteActive) begin
        if (BitCounter == 8) begin
            if (WrCycle | (START_Rcvd & (ShiftRegister[07:01] == {4'b1010,ChipAddress[02:00]}))) begin
                SDA_DO <= 0;
                SDA_OE <= 1;
            end
        end
    end
    if (BitCounter == 9) begin
        BitCounter <= 0;

        if (!RdCycle) begin
            SDA_DO <= 0;
            SDA_OE <= 0;
        end
    end
end

```

```

        end
    end
end

// -----
//      1.09:  Acknowledge Detect
// -----

always @(posedge SCL) begin
    if (RdCycle & (BitCounter == 8)) begin
        if ((SDA == 0) & (SDA_OE == 0)) MACK_Rcvd <= 1;
    end
end

always @(negedge SCL) MACK_Rcvd <= 0;

// -----
//      1.10:  Write Cycle Timer
// -----

always @(posedge STOP_Rcvd) begin
    if (WrCycle & (WP == 0) & (WrCounter > 0)) begin
        WriteActive = 1;
        #(tWC);
        WriteActive = 0;
    end
end

always @(posedge STOP_Rcvd) begin
    #(1.0);
    STOP_Rcvd = 0;
end

// -----
//      1.11:  Write Cycle Processor
// -----

always @(posedge WriteActive) begin
    for (LoopIndex = 0; LoopIndex < WrCounter; LoopIndex = LoopIndex + 1) begin
        PageAddress = StartAddress[04:00] + LoopIndex;

        case (StartAddress[12:10])
            3'b000 : MemoryBlock0[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b001 : MemoryBlock1[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b010 : MemoryBlock2[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b011 : MemoryBlock3[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b100 : MemoryBlock4[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b101 : MemoryBlock5[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b110 : MemoryBlock6[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
            3'b111 : MemoryBlock7[{StartAddress[09:05],PageAddress[04:00]}] = WrDataByte[LoopIndex[04:00]];
        endcase
    end
end

// -----
//      1.12:  Read Data Multiplexor
// -----

always @(negedge SCL) begin
    if (BitCounter == 8) begin
        if (WrCycle & ADLO_Rcvd) begin
            RdPointer <= StartAddress + WrPointer + 1;
        end
        if (RdCycle) begin
            RdPointer <= RdPointer + 1;
        end
    end
end

assign RdDataByte = {8{(RdPointer[12:10] == 0)}} & MemoryBlock0[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 1)}} & MemoryBlock1[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 2)}} & MemoryBlock2[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 3)}} & MemoryBlock3[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 4)}} & MemoryBlock4[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 5)}} & MemoryBlock5[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 6)}} & MemoryBlock6[RdPointer[09:00]]
    | {8{(RdPointer[12:10] == 7)}} & MemoryBlock7[RdPointer[09:00]];

// -----
//      1.13:  Read Data Processor
// -----

```

```

always @(negedge SCL) begin
  if (RdCycle) begin
    if (BitCounter == 8) begin
      SDA_DO <= 0;
      SDA_OE <= 0;
    end
    else if (BitCounter == 9) begin
      SDA_DO <= RdDataByte[07];

      if (MACK_Rcvd) SDA_OE <= 1;
    end
    else begin
      SDA_DO <= RdDataByte[7-BitCounter];
    end
  end
end

// -----
// 1.14: SDA Data I/O Buffer
// -----

bufif1 (SDA, 1'b0, SDA_DriveEnableDlyd);

assign SDA_DriveEnable = !SDA_DO & SDA_OE;
always @(SDA_DriveEnable) SDA_DriveEnableDlyd <= #(tAA) SDA_DriveEnable;

// *****
// ** DEBUG LOGIC **
// *****
// -----
// 2.01: Memory Data Bytes
// -----

wire [07:00] MemoryByte0_000 = MemoryBlock0[00];
wire [07:00] MemoryByte0_001 = MemoryBlock0[01];
wire [07:00] MemoryByte0_002 = MemoryBlock0[02];
wire [07:00] MemoryByte0_003 = MemoryBlock0[03];
wire [07:00] MemoryByte0_004 = MemoryBlock0[04];
wire [07:00] MemoryByte0_005 = MemoryBlock0[05];
wire [07:00] MemoryByte0_006 = MemoryBlock0[06];
wire [07:00] MemoryByte0_007 = MemoryBlock0[07];
wire [07:00] MemoryByte0_008 = MemoryBlock0[08];
wire [07:00] MemoryByte0_009 = MemoryBlock0[09];
wire [07:00] MemoryByte0_00A = MemoryBlock0[10];
wire [07:00] MemoryByte0_00B = MemoryBlock0[11];
wire [07:00] MemoryByte0_00C = MemoryBlock0[12];
wire [07:00] MemoryByte0_00D = MemoryBlock0[13];
wire [07:00] MemoryByte0_00E = MemoryBlock0[14];
wire [07:00] MemoryByte0_00F = MemoryBlock0[15];

wire [07:00] MemoryByte1_000 = MemoryBlock1[00];
wire [07:00] MemoryByte1_001 = MemoryBlock1[01];
wire [07:00] MemoryByte1_002 = MemoryBlock1[02];
wire [07:00] MemoryByte1_003 = MemoryBlock1[03];
wire [07:00] MemoryByte1_004 = MemoryBlock1[04];
wire [07:00] MemoryByte1_005 = MemoryBlock1[05];
wire [07:00] MemoryByte1_006 = MemoryBlock1[06];
wire [07:00] MemoryByte1_007 = MemoryBlock1[07];
wire [07:00] MemoryByte1_008 = MemoryBlock1[08];
wire [07:00] MemoryByte1_009 = MemoryBlock1[09];
wire [07:00] MemoryByte1_00A = MemoryBlock1[10];
wire [07:00] MemoryByte1_00B = MemoryBlock1[11];
wire [07:00] MemoryByte1_00C = MemoryBlock1[12];
wire [07:00] MemoryByte1_00D = MemoryBlock1[13];
wire [07:00] MemoryByte1_00E = MemoryBlock1[14];
wire [07:00] MemoryByte1_00F = MemoryBlock1[15];

wire [07:00] MemoryByte2_000 = MemoryBlock2[00];
wire [07:00] MemoryByte2_001 = MemoryBlock2[01];
wire [07:00] MemoryByte2_002 = MemoryBlock2[02];
wire [07:00] MemoryByte2_003 = MemoryBlock2[03];
wire [07:00] MemoryByte2_004 = MemoryBlock2[04];
wire [07:00] MemoryByte2_005 = MemoryBlock2[05];
wire [07:00] MemoryByte2_006 = MemoryBlock2[06];
wire [07:00] MemoryByte2_007 = MemoryBlock2[07];
wire [07:00] MemoryByte2_008 = MemoryBlock2[08];
wire [07:00] MemoryByte2_009 = MemoryBlock2[09];
wire [07:00] MemoryByte2_00A = MemoryBlock2[10];
wire [07:00] MemoryByte2_00B = MemoryBlock2[11];
wire [07:00] MemoryByte2_00C = MemoryBlock2[12];
wire [07:00] MemoryByte2_00D = MemoryBlock2[13];

```



```

wire [07:00] MemoryByte7_00B = MemoryBlock7[11];
wire [07:00] MemoryByte7_00C = MemoryBlock7[12];
wire [07:00] MemoryByte7_00D = MemoryBlock7[13];
wire [07:00] MemoryByte7_00E = MemoryBlock7[14];
wire [07:00] MemoryByte7_00F = MemoryBlock7[15];

// -----
//      2.02: Write Data Buffer
// -----

wire [07:00] WriteData_00 = WrDataByte[00];
wire [07:00] WriteData_01 = WrDataByte[01];
wire [07:00] WriteData_02 = WrDataByte[02];
wire [07:00] WriteData_03 = WrDataByte[03];
wire [07:00] WriteData_04 = WrDataByte[04];
wire [07:00] WriteData_05 = WrDataByte[05];
wire [07:00] WriteData_06 = WrDataByte[06];
wire [07:00] WriteData_07 = WrDataByte[07];
wire [07:00] WriteData_08 = WrDataByte[08];
wire [07:00] WriteData_09 = WrDataByte[09];
wire [07:00] WriteData_0A = WrDataByte[10];
wire [07:00] WriteData_0B = WrDataByte[11];
wire [07:00] WriteData_0C = WrDataByte[12];
wire [07:00] WriteData_0D = WrDataByte[13];
wire [07:00] WriteData_0E = WrDataByte[14];
wire [07:00] WriteData_0F = WrDataByte[15];

wire [07:00] WriteData_10 = WrDataByte[16];
wire [07:00] WriteData_11 = WrDataByte[17];
wire [07:00] WriteData_12 = WrDataByte[18];
wire [07:00] WriteData_13 = WrDataByte[19];
wire [07:00] WriteData_14 = WrDataByte[20];
wire [07:00] WriteData_15 = WrDataByte[21];
wire [07:00] WriteData_16 = WrDataByte[22];
wire [07:00] WriteData_17 = WrDataByte[23];
wire [07:00] WriteData_18 = WrDataByte[24];
wire [07:00] WriteData_19 = WrDataByte[25];
wire [07:00] WriteData_1A = WrDataByte[26];
wire [07:00] WriteData_1B = WrDataByte[27];
wire [07:00] WriteData_1C = WrDataByte[28];
wire [07:00] WriteData_1D = WrDataByte[29];
wire [07:00] WriteData_1E = WrDataByte[30];
wire [07:00] WriteData_1F = WrDataByte[31];

// *****
// **      TIMING CHECKS      **
// *****

wire TimingCheckEnable = (RESET == 0) & (SDA_OE == 0);

specify
  specparam
    tHI = 600,           // SCL pulse width - high
    tLO = 1300,         // SCL pulse width - low
    tSU_STA = 600,      // SCL to SDA setup time
    tHD_STA = 600,      // SCL to SDA hold time
    tSU_DAT = 100,      // SDA to SCL setup time
    tSU_STO = 600;      // SCL to SDA setup time

    $width (posedge SCL, tHI);
    $width (negedge SCL, tLO);

    $setup (SCL, negedge SDA &&& TimingCheckEnable, tSU_STA);
    $setup (SDA, posedge SCL &&& TimingCheckEnable, tSU_DAT);
    $setup (SCL, posedge SDA &&& TimingCheckEnable, tSU_STO);

    $hold (negedge SDA &&& TimingCheckEnable, SCL, tHD_STA);
endspecify
endmodule

```