

```

// *****
// **
// ** M24LC00.v - 24LC00 128-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      : 1.0
// ** Modified Date : 09/10/2004
// ** Revision History:
// **
// ** 09/10/2004: Initial design
// **
// *****
// **          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
// **
// **-----**
// ** TIMING CHECKS
// **-----**
// **
// *****

```

`timescale 1ns/10ps

module M24LC00 (A0, A1, A2, WP, SDA, SCL, RESET);

```

input      A0;           // unconnected pin
input      A1;           // unconnected pin
input      A2;           // unconnected pin

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_DriveEnabledDlyd;     // serial data output enable - delayed

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           ADDR_Rcvd;                 // byte address 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 [03:00]   StartAddress;              // memory access starting address
reg [02:00]   PageAddress;              // memory page address

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

reg [15:00]   WrCounter;                 // write buffer counter

reg [03:00]   RdPointer;                 // read address pointer

reg           WriteActive;               // memory write cycle active

reg [07:00]   MemoryBlock [0:15];       // EEPROM data memory array

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;
    ADDR_Rcvd = 0;
    MACK_Rcvd = 0;
end

initial begin
    BitCounter = 0;
    ControlByte = 0;
end

initial begin
    WrCycle = 0;
    RdCycle = 0;

    WriteActive = 0;
end

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

always @(negedge SDA) begin
    if (SCL == 1) begin
        START_Rcvd <= 1;
        STOP_Rcvd <= 0;
    end
end

```

```

CTRL_Rcvd  <= 0;
ADDR_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;
        ADDR_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[07:04] == 4'b1010)) begin
            if (ShiftRegister[00] == 0) WrCycle <= 1;
            if (ShiftRegister[00] == 1) RdCycle <= 1;

            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 <= ShiftRegister[03:00];
        end
    end
end

```

```

        RdPointer    <= ShiftRegister[03:00];

        ADDR_Rcvd <= 1;
    end

    WrCounter <= 0;

    CTRL_Rcvd <= 0;
end
end

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

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

            WrCounter <= 1;
        end
    end
end

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

always @(negedge SCL) begin
    if (!WriteActive) begin
        if (BitCounter == 8) begin
            if (WrCycle | (START_Rcvd & (ShiftRegister[07:04] == 4'b1010))) begin
                SDA_DO <= 0;
                SDA_OE <= 1;
            end
        end
        if (BitCounter == 9) begin
            BitCounter <= 0;

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

```

```

MemoryBlock[StartAddress[03:00]] = WrDataByte;
end

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

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

assign RdDataByte = MemoryBlock[RdPointer[03:00]];

// -----
// 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] MemoryByte00 = MemoryBlock[00];
wire [07:00] MemoryByte01 = MemoryBlock[01];
wire [07:00] MemoryByte02 = MemoryBlock[02];
wire [07:00] MemoryByte03 = MemoryBlock[03];
wire [07:00] MemoryByte04 = MemoryBlock[04];
wire [07:00] MemoryByte05 = MemoryBlock[05];
wire [07:00] MemoryByte06 = MemoryBlock[06];
wire [07:00] MemoryByte07 = MemoryBlock[07];

wire [07:00] MemoryByte08 = MemoryBlock[08];
wire [07:00] MemoryByte09 = MemoryBlock[09];
wire [07:00] MemoryByte0A = MemoryBlock[10];
wire [07:00] MemoryByte0B = MemoryBlock[11];
wire [07:00] MemoryByte0C = MemoryBlock[12];
wire [07:00] MemoryByte0D = MemoryBlock[13];
wire [07:00] MemoryByte0E = MemoryBlock[14];
wire [07:00] MemoryByte0F = MemoryBlock[15];

// *****
// ** 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
```