

```

/*
*****

*   File Name: w25p022a.v
*   Product:   W25P022A
*   Features:  64k x 32 burst pipelined SRAM
*               access time: 6ns
*               pipelined data output capability
*               2T/1T mode Signal cycle deslected mode
*               Intel burst mode & linear burst mode selection (LBO)
*
*   Date : December 21, 1997
*           Written by Lin, Hung-Hsueh
*           Email Address : hhlin@winbond.com.tw
*
*   Verilog model for 62kx32 Burst Pipelined SRAM
*   (Tested on Verilog-XL 2.2.27)
*
*
*   Memory Product Dept.(PM20)
*   Winbond Electronics Crop.
*
*   Copyright (c) 1997 Winbond Electronics Corp.
*   All right reserved
*
*   -----
*   Version 1.1 January 14 , 1998
*   (addr_reg_out type)
*****

*/

`timescale 1ns/100ps
module pbsram (addr, io, clk, ce_1, ce2, ce_3, gw, bwe, bw1, bw2, bw3, bw4, oe, adv,
adsc, adsp, zz, ft, lbo);

parameter maxDepth = 65536;    //64k
parameter maxAddr = 16;        //Address
parameter maxOut = 32;         //IO
parameter maxbyte = 4;

```

```
parameter reg_delay = 0.3;
```

```
inout [maxOut-1:0] io;
```

```
input [maxAddr-1:0] addr;
```

```
input clk, ce_1, ce2, ce_3;
```

```
input adv, adsc, adsp;
```

```
input gw, bwe, bw1, bw2, bw3, bw4, oe;
```

```
input zz , ft, lbo;
```

```
reg enable, pipe_enable;
```

```
reg wr_reg_out1, wr_reg_out2, wr_reg_out3, wr_reg_out4;
```

```
reg [1:0] count;
```

```
reg [maxAddr-1:0] addr_reg_in;
```

```
reg [maxOut-1 :0] dout, data_out;
```

```
reg [maxOut-1:0] din;
```

```
reg [(maxOut/4-1):0] array_quad1 [0:maxDepth];
```

```
reg [(maxOut/4-1):0] array_quad2 [0:maxDepth];
```

```
reg [(maxOut/4-1):0] array_quad3 [0:maxDepth];
```

```
reg [(maxOut/4-1):0] array_quad4 [0:maxDepth];
```

```
wire en_int = (~ce_1 & ce2 & ~ce_3);
```

```
wire ce_adsp = (~adsp & ~ce_1);
```

```
wire load = (ce_adsp | ~adsc) ;
```

```
wire addbit1, addbit0;
```

```
wire [maxAddr-1:0] addr_reg_out;
```

```
wire bw_en1 = (bw1 | bwe) & gw;
```

```
wire bw_en2 = (bw2 | bwe) & gw;
```

```
wire bw_en3 = (bw3 | bwe) & gw;
```

```
wire bw_en4 = (bw4 | bwe) & gw;
```

```
wire wr_reg_or = wr_reg_out1 | wr_reg_out2 | wr_reg_out3 | wr_reg_out4;
```

```
wire oe_en;
```

```
wire [maxOut-1:0] io;
```

```
wire [maxOut-1:0] io_int;
```

```

integer i;

initial
begin
    enable =0;
    $timeformat(-9,1,"ns",10);
    for (i=0; i<=maxDepth; i=i+1)
        begin
            array_quad1[i] =8'b0;
            array_quad2[i] =8'b0;
            array_quad3[i] =8'b0;
            array_quad4[i] =8'b0;
        end
end

// address register
always @(posedge clk)
begin
    if (load)
        addr_reg_in <= #0.2 addr;
end

// burst counter
always @(posedge clk)
begin
    if (~lbo & load)
        count <= 2'b00;
    else
        if (lbo & load)
            count <= addr [1:0];
        else
            if (~adv & ~load)
                count = count + 1;
end

assign addbit1 = lbo ? count[1] : (count[1]^addr_reg_in[1]) ;
assign addbit0 = lbo ? count[0] : (count[0]^addr_reg_in[0]) ;
assign addr_reg_out = {addr_reg_in[maxAddr-1:2], addbit1, addbit0};

```

```

// write register
always @(posedge clk)
begin
    wr_reg_out1 <= #0.2 ~(ce_adsp || bw_en1);
    wr_reg_out2 <= #0.2 ~(ce_adsp || bw_en2);
    wr_reg_out3 <= #0.2 ~(ce_adsp || bw_en3);
    wr_reg_out4 <= #0.2 ~(ce_adsp || bw_en4);
end

// enable register
always @(posedge clk)
begin
    if (load)
        enable = #0.2 en_int;
end

// pipelined enable
always @(posedge clk)
begin
    pipe_enable <= #0.2 enable;
end

// memory array
always @(posedge clk)
begin
    if (en_int & ~adsc & adsp)
    begin
        #1.0
        if (~bw_en1) array_quad1[addr] <= din[7:0];
        if (~bw_en2) array_quad2[addr] <= din[15:8];
        if (~bw_en3) array_quad3[addr] <= din[23:16];
        if (~bw_en4) array_quad4[addr] <= din[31:24];
    end
    else if (enable & (adsc & (adsp | ce_1)))
    begin
        #1.0;
        if (~bw_en1) array_quad1[addr_reg_out] <= din[7:0];
    end
end

```

```

        if (~bw_en2) array_quad2[addr_reg_out] <= din[15:8];
        if (~bw_en3) array_quad3[addr_reg_out] <= din[23:16];
        if (~bw_en4) array_quad4[addr_reg_out] <= din[31:24];
    end

end

// input register
always @(posedge clk)
begin
    din <= #0.3 io;
end

// output register
always @(posedge clk)
begin
    #0.2;
    #0.2;
    if ( ~wr_reg_or)
    begin
        dout[7 : 0] <= #0.2 array_quad1 [addr_reg_out];
        dout[15: 8] <= #0.2 array_quad2 [addr_reg_out];
        dout[23:16] <= #0.2 array_quad3 [addr_reg_out];
        dout[31:24] <= #0.2 array_quad4 [addr_reg_out];
    end
end

end

always @(posedge clk)
begin
    #0.2;
    if (wr_reg_or)
        data_out <= din ;
    else if ( ~wr_reg_or)
        data_out <= dout ;
end

assign io_int =  oe_en ? data_out : din ;

```

```
assign oe_en = (~oe & pipe_enable & enable & ~wr_reg_or);
```

```
//output driver
```

```
bufifl #0.2 (io[0], io_int[0], oe_en);  
bufifl #0.2 (io[1], io_int[1], oe_en);  
bufifl #0.2 (io[2], io_int[2], oe_en);  
bufifl #0.2 (io[3], io_int[3], oe_en);  
bufifl #0.2 (io[4], io_int[4], oe_en);  
bufifl #0.2 (io[5], io_int[5], oe_en);  
bufifl #0.2 (io[6], io_int[6], oe_en);  
bufifl #0.2 (io[7], io_int[7], oe_en);  
bufifl #0.2 (io[8], io_int[8], oe_en);  
bufifl #0.2 (io[9], io_int[9], oe_en);  
bufifl #0.2 (io[10], io_int[10], oe_en);  
bufifl #0.2 (io[11], io_int[11], oe_en);  
bufifl #0.2 (io[12], io_int[12], oe_en);  
bufifl #0.2 (io[13], io_int[13], oe_en);  
bufifl #0.2 (io[14], io_int[14], oe_en);  
bufifl #0.2 (io[15], io_int[15], oe_en);  
bufifl #0.2 (io[16], io_int[16], oe_en);  
bufifl #0.2 (io[17], io_int[17], oe_en);  
bufifl #0.2 (io[18], io_int[18], oe_en);  
bufifl #0.2 (io[19], io_int[19], oe_en);  
bufifl #0.2 (io[20], io_int[20], oe_en);  
bufifl #0.2 (io[21], io_int[21], oe_en);  
bufifl #0.2 (io[22], io_int[22], oe_en);  
bufifl #0.2 (io[23], io_int[23], oe_en);  
bufifl #0.2 (io[24], io_int[24], oe_en);  
bufifl #0.2 (io[25], io_int[25], oe_en);  
bufifl #0.2 (io[26], io_int[26], oe_en);  
bufifl #0.2 (io[27], io_int[27], oe_en);  
bufifl #0.2 (io[28], io_int[28], oe_en);  
bufifl #0.2 (io[29], io_int[29], oe_en);  
bufifl #0.2 (io[30], io_int[30], oe_en);  
bufifl #0.2 (io[31], io_int[31], oe_en);
```

```
specify
```

```

specparam tCYC=13.3, //Clock Cycle time
    tKH=5,           //Colok High Pulse Width
    tKL=5,           //Colok Low Pulse Width
    tKQ=6,           //Clock to Output Valid
    tKX=2,           //Clock to Output Invalid
    tKHZ=6.5,        //Clock to Output High-Z
    tKLZ=0,          //Clock to Output Low-Z
    tOE=6,           //Output Enable to Output Valid
    tOHZ=6,          //Output Enable to High-Z
    tOLZ=0,          //Output Enable to Low-Z
    tAS=2.5, //Add. Setup Time
    tADSS=2.5, //ADSP_, ADSC_, ADV_ Setup Time
    tCES=2.5, //CE_1_, CE2, CE_3_ Setup Time
    tWS=2.5, //GW_, BWE_ BWx_ Setup Time
    tDS=2.5, //Write Data Setup Time
    tAH=0.5, //Add. Hold Time
    tADSH=0.5, //ADSP_, ADSC_, ADV_ Hold Time
    tCEH=0.5, //CE_1_, CE2, CE_3_ Hold Time
    tWH=0.5, //GW_, BWE_ BWx_ Hold Time
    tDH=0.5; //Write Data Hold Time

```

```

(oe *>io)=(tOE,tOE,tOHZ);

```

```

(clk *>io)=(tKQ,tKQ,tKHZ);

```

```

$width (posedge clk, tKH);
$width (negedge clk, tKL);
$period (posedge clk, tCYC);
$period (negedge clk, tCYC);
$setuphold(posedge clk, adsc, tADSS, tADSH);
$setuphold(posedge clk, adsp, tADSS, tADSH);
$setuphold(posedge clk, adv, tADSS, tADSH);
$setuphold(posedge clk, bw1, tWS, tWH);
$setuphold(posedge clk, bw2, tWS, tWH);
$setuphold(posedge clk, bw3, tWS, tWH);
$setuphold(posedge clk, bw4, tWS, tWH);
$setuphold(posedge clk, gw, tWS, tWH);
$setuphold(posedge clk, bwe, tWS, tWH);
$setuphold(posedge clk, ce_1, tCES, tCEH);

```

```
    $setuphold(posedge clk, ce2, tCES, tCEH);  
    $setuphold(posedge clk, ce_3, tCES, tCEH);  
    $setuphold(posedge clk, addr, tAS, tAH);  
endspecify  
  
endmodule
```