





//----------fifo\_interface.v----------

`define SRAM\_SIZE 8 //为减小对FIFO控制器的测试工作量，置SRAM空间为8字节

`timescale 1ns/1ns

module fifo\_interface(

in\_data, //用户的输入数据总线

out\_data, //用户的输出数据总线

fiford, //FIFO读控制信号，低电平有效

fifowr, //FIFO写控制信号，低电平有效

nfull,

nempty,

address, //到SRAM的地址总线

sram\_data, //到SRAM的双向数据总线

rd, //SRAM读使能，低电平有效

wr, //SRAM写使能，低电平有效

clk, //系统时钟信号

rst); //全局复位信号，低电平有效

//来自用户的控制输入信号

input fiford,fifowr,clk,rst;

//来自用户的数据信号

input[7:0] in\_data;

output[7:0] out\_data;

reg[7:0] in\_data\_buf, //输入数据缓冲区

out\_data\_buf; //输出数据缓冲区

//输出到用户的状态指示信号

output nfull,nempty;

reg nfull,nempty;

//输出到SRAM的控制信号

output rd,wr;

//到SRAM的双向数据总线

inout[7:0] sram\_data;

//输出到SRAM的地址总线

output[10:0] address;

reg[10:0] address;

//Internal Register

reg[10:0] fifo\_wp, //FIFO写指针

fifo\_rp; //FIFO读指针

reg[10:0] fifo\_wp\_next, //fifo\_wp的下一个值

fifo\_rp\_next; //fifo\_rp的下一个值

reg near\_full,near\_empty;

reg[3:0] state; //SRAM操作状态机寄存器

parameter idle='b0000,read\_ready='b0100,read='b0101,read\_over='b0111,

write\_ready='b1000,write='b1001,write\_over='b1011;

//SRAM操作状态机

always@(posedge clk or negedge rst)

if(~rst)

state<=idle;

else

case(state)

idle: //等待FIFO的操作控制信号

if(fifowr==0&&nfull) //用户发出写FIFO申请，且FIFO未满

state<=write\_ready;

else if(fiford==0&&nempty) //用户发出读FIFO申请，且FIFO未空

state<=read\_ready;

else //没有对FIFO操作的申请

state<=idle;

read\_ready: //建立SRAM操作所需地址和数据

state<=read;

read: //等待用户结束当前读操作

if(fiford==1)

state<=read\_over;

else

state<=read;

read\_over: //继续给出SRAM地址以保证数据稳定

state<=idle;

write\_ready: //建立SRAM操作所需地址和数据

state<=write;

write: //等待用户结束当前写操作

if(fifowr==1)

state<=write\_over;

else

state<=write;

write\_over: //继续给出SRAM地址和写入数据以保证数据稳定

state<=idle;

default: state<=idle;

endcase

//产生SRAM操作相关信号

assign rd = ~state[2];

assign wr = (state==write)?fifowr:1'b1;

always@(posedge clk)

if(~fifowr)

in\_data\_buf<=in\_data;

assign sram\_data=(state[3])?in\_data\_buf:8'hzz;

always@(state or fiford or fifowr or fifo\_wp or fifo\_rp)

if(state[2]||~fiford)

address=fifo\_rp;

else if(state[3]||~fifowr)

address=fifo\_wp;

else

address='bz;

//产生FIFO数据

assign out\_data = (state[2])?sram\_data:8'bz;

always@(posedge clk)

if(state==read)

out\_data\_buf<=sram\_data;

//计算FIFO读写指针

always@(posedge clk or negedge rst)

if(~rst)

fifo\_rp<=0;

else if(state==read\_over)

fifo\_rp<=fifo\_rp\_next;

always@(fifo\_rp)

if(fifo\_rp==`SRAM\_SIZE-1)

fifo\_rp\_next=0;

else

fifo\_rp\_next=fifo\_rp+1;

always@(posedge clk or negedge rst)

if(~rst)

fifo\_wp<=0;

else if(state==write\_over)

fifo\_wp<=fifo\_wp\_next;

always@(fifo\_wp)

if(fifo\_wp==`SRAM\_SIZE-1)

fifo\_wp\_next=0;

else

fifo\_wp\_next=fifo\_wp+1;

always@(posedge clk or negedge rst)

if(~rst)

near\_empty<=1'b0;

else if(fifo\_wp==fifo\_rp\_next)

near\_empty<=1'b1;

else

near\_empty<=1'b0;

always@(posedge clk or negedge rst)

if(~rst)

nempty<=1'b0;

else if(near\_empty&&state==read)

nempty<=1'b0;

else if(state==write)

nempty<=1'b1;

always@(posedge clk or negedge rst)

if(~rst)

near\_full<=1'b0;

else if(fifo\_rp==fifo\_wp\_next)

near\_full<=1'b1;

else

near\_full<=1'b0;

always@(posedge clk or negedge rst)

if(~rst)

nfull<=1'b1;

else if(near\_full&&state==write)

nfull<=1'b0;

else if(state==read)

nfull<=1'b1;

endmodule

//----------sram.v----------

module sram(Address,Data,SRG,SRE,SRW);

input[10:0] Address;

input SRG, //Output enable

SRE, //Chip enable

SRW; //Write enable

inout[7:0] Data; //Bus

wire[10:0] Addr=Address;

reg[7:0] RdData;

reg[7:0] SramMem[0:'h7ff];

reg RdSramDly,RdFlip;

wire[7:0] FlpData,Data;

reg WR\_flag; //To judge the signals according to the specification of HM-65162

integer i;

wire RdSram=~SRG&~SRE;

wire WrSram=~SRW&~SRE;

reg[10:0] DelayAddr;

reg[7:0] DelayData;

reg WrSramDly;

integer file;

assign FlpData = (RdFlip)?~RdData:RdData;

assign Data = (RdSramDly)?FlpData:'hz;

parameter TAVQV=90, //2 (max) Address access time

TELQV=90, //3 (max) Chip enable access time

TELQX=5, //4 (min) Chip enable output enable time

TGLQV=65, //5 (max) Output enable access time

TGLQX=5, //6 (min) Output enable output enable time

TEHQZ=50, //7 (max) Chip enable output disable time

TGHQZ=40, //8 (max) Output enable output disable time

TAVQX=5; //9 (min) Output hold from address change

parameter TAVWL=10, //12 (min) Address setup time

TWLWH=55, //13 (min) Chip enable pulse setup time

TWHAX=15, //14 (min10) Write enable read setup time

TWLQZ=50, //16 (max) Write enable output disable time

TDVWH=30, //17 (min) Data setup time

TWHDX=20, //18 (min15) Data hold time

TWHQX=20, //19 (min0) Write enable output enable time

TWLEH=55, //20 (min) Write enable pulse setup time

TDVEH=30, //21 (min) Chip enable data setup time

TAVWH=70; //22 (min65) Address valid to end of write

initial

begin

file=$fopen("ramlow.txt");

if(!file)

begin

$display("Could not open the file.");

$stop;

end

end

initial

begin

for(i=0;i<'h7ff;i=i+1)

SramMem[i]=i;

$monitor($time,"DelayAddr=%h,DelayData=%h",DelayAddr,DelayData);

end

initial RdSramDly=0;

initial WR\_flag=1;

always@(posedge RdSram) #TGLQX RdSramDly=RdSram;

always@(posedge SRW) #TWHQX RdSramDly=RdSram;

always@(Addr)

begin

#TAVQX;

RdFlip=1;

#(TGLQV-TAVQX)

if(RdSram)

RdFlip=0;

end

always@(posedge RdSram)

begin

RdFlip=1;

#TAVQV;

if(RdSram)

RdFlip=0;

end

always@(Addr) #TAVQX RdFlip=1;

always@(posedge SRG) #TEHQZ RdSramDly=RdSram;

always@(posedge SRE) #TGHQZ RdSramDly=RdSram;

always@(negedge SRW) #TWLQZ RdSramDly=0;

always@(negedge WrSramDly or posedge RdSramDly) RdData=SramMem[Addr];

always@(Addr) #TAVWL DelayAddr=Addr;

always@(Data) #TDVWH DelayData=Data;

always@(WrSram) #5 WrSramDly=WrSram;

always@(Addr or Data or WrSram) WR\_flag=1;

always@(negedge SRW)

begin

#TWLWH;

if(SRW)

begin

WR\_flag=0;

$display("ERROR!Can't write!Write enable time(W) is too short!");

end

end

always@(posedge SRW)

begin

#TWHAX;

if(DelayAddr!==Addr)

begin

WR\_flag=0;

$display("ERROR!Can't write!Write enable read setup time is too short!");

end

end

always@(Data)

if(WrSram)

begin

#TDVEH;

if(SRE)

begin

WR\_flag=0;

$display("ERROR!Can't write!Chip enable data setup time is too short!");

end

end

always@(Data)

if(WrSram)

begin

#TDVEH;

if(SRW)

begin

WR\_flag=0;

$display("ERROR!Can't write!Chip enable data setup time is too short!");

end

end

always@(posedge SRW)

begin

#TWHDX

if(DelayData!==Data)

$display("Warning!Data hold time is too short!");

end

always@(DelayAddr or DelayData or WrSramDly)

if(WrSram&&WR\_flag)

begin

if(!Addr[5])

begin

#15 SramMem[Addr]=Data;

$display("mem[%h]=%h",Addr,Data);

$fwrite(file,"mem[%h]=%h",Addr,Data);

if(Addr[0]&&Addr[1])

$fwrite(file,"\n");

end

else

begin

$fclose(file);

$display("Please check the txt.");

$stop;

end

end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 20:38:11 05/22/2023

// Design Name: fifo\_interface

// Module Name: D:/ssss522/t.v

// Project Name: ssss522

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: fifo\_interface

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

////////////////////////////////////////////////////////////////////////////////

//----------fifo\_interface.vt----------

`define FIFO\_SIZE 8

`timescale 1 ns/ 1 ns

module fifo\_interface\_vlg\_tst();

reg[7:0] in\_data;

reg fiford,fifowr;

wire[7:0] out\_data;

wire nfull,nempty;

reg clk,rst;

wire[7:0] sram\_data;

wire[10:0] address;

wire rd,wr;

reg[7:0] data\_buf[`FIFO\_SIZE:0];

integer index;

//系统时钟

initial clk=0;

always #25 clk=~clk;

//测试激励序列

initial

begin

fiford=1;

fifowr=1;

rst=1;

#40 rst=0;

#42 rst=1;

if(nempty)

$display($time,"Error:FIFO be empty, nempty should be low.\n");

index=0;

repeat(`FIFO\_SIZE)

begin

data\_buf[index]=$random;

write\_fifo(data\_buf[index]);

index=index+1;

end

if(nfull)

$display($time,"Error:FIFO full, nfull should be low.\n");

repeat(2) write\_fifo($random);

#200

index=0;

read\_fifo\_compare(data\_buf[index]);

if(~nfull)

$display($time,"Error:FIFO not full, nfull should be high.\n");

repeat(`FIFO\_SIZE-1)

begin

index=index+1;

read\_fifo\_compare(data\_buf[index]);

end

if(nempty)

$display($time,"Error:FIFO be empty, nempty should be low.\n");

repeat(2) read\_fifo\_compare(8'bx);

reset\_fifo;

repeat(`FIFO\_SIZE\*2)

begin

data\_buf[0]=$random;

write\_fifo(data\_buf[0]);

read\_fifo\_compare(data\_buf[0]);

end

reset\_fifo;

read\_fifo\_compare(8'bx);

write\_fifo(data\_buf[0]);

read\_fifo\_compare(data\_buf[0]);

$stop;

end

fifo\_interface i1 (

.address(address),

.clk(clk),

.fiford(fiford),

.fifowr(fifowr),

.in\_data(in\_data),

.nempty(nempty),

.nfull(nfull),

.out\_data(out\_data),

.rd(rd),

.rst(rst),

.sram\_data(sram\_data),

.wr(wr)

);

sram m1(.Address(address),.Data(sram\_data),.SRG(rd),.SRE(1'b0),.SRW(wr));

task write\_fifo;

input[7:0] data;

begin

in\_data=data;

#50 fifowr=0;

#200 fifowr=1;

#50;

end

endtask

task read\_fifo\_compare;

input[7:0] data;

begin

#50 fiford=0;

#200 fiford=1;

if(out\_data!=data)

$display($time,"Error:Data retrieved(%h)not match the one stored(%h).\n",out\_data,data);

#50;

end

endtask

task reset\_fifo;

begin

#40 rst=0;

#40 rst=1;

end

endtask

endmodule