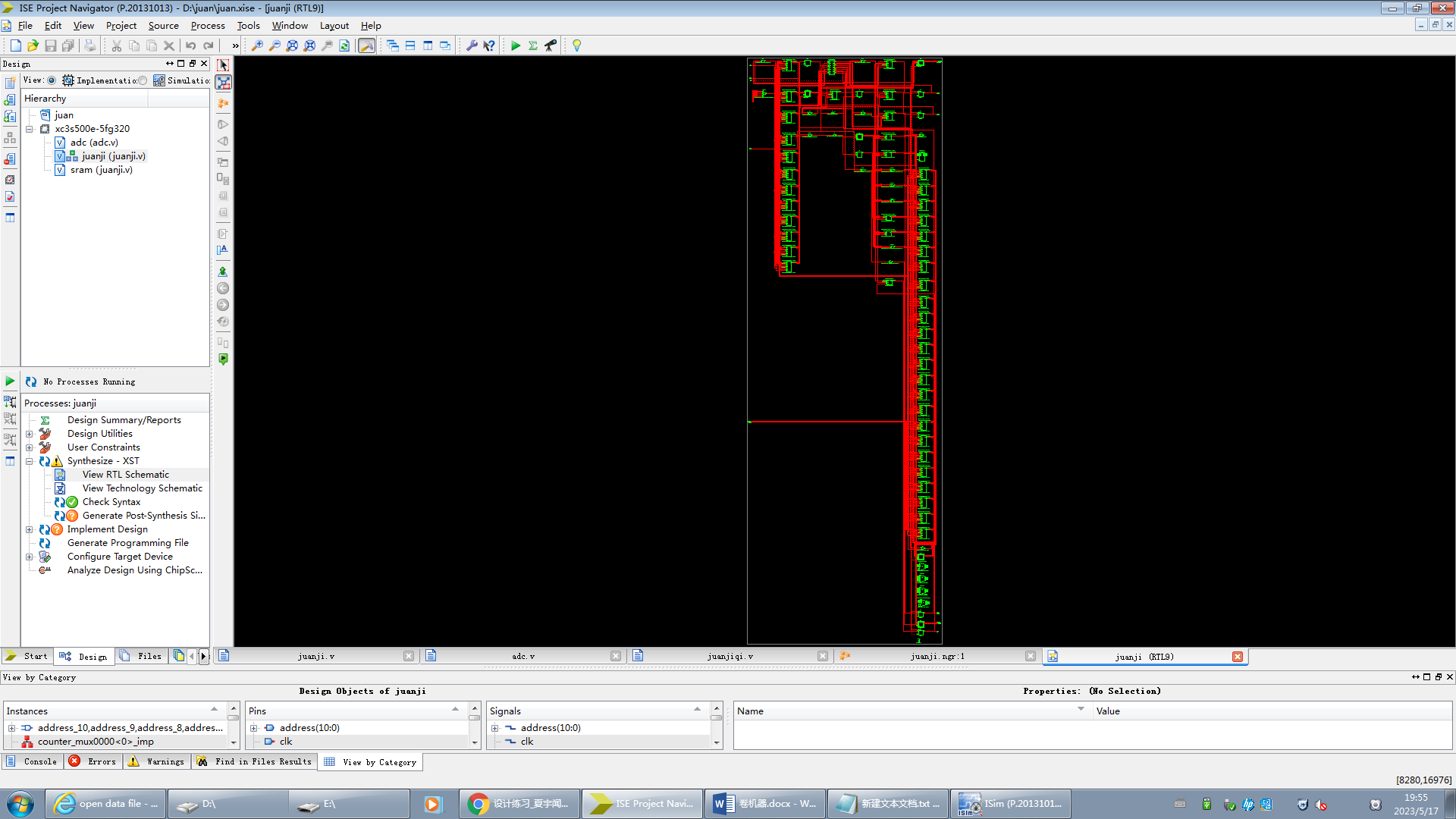
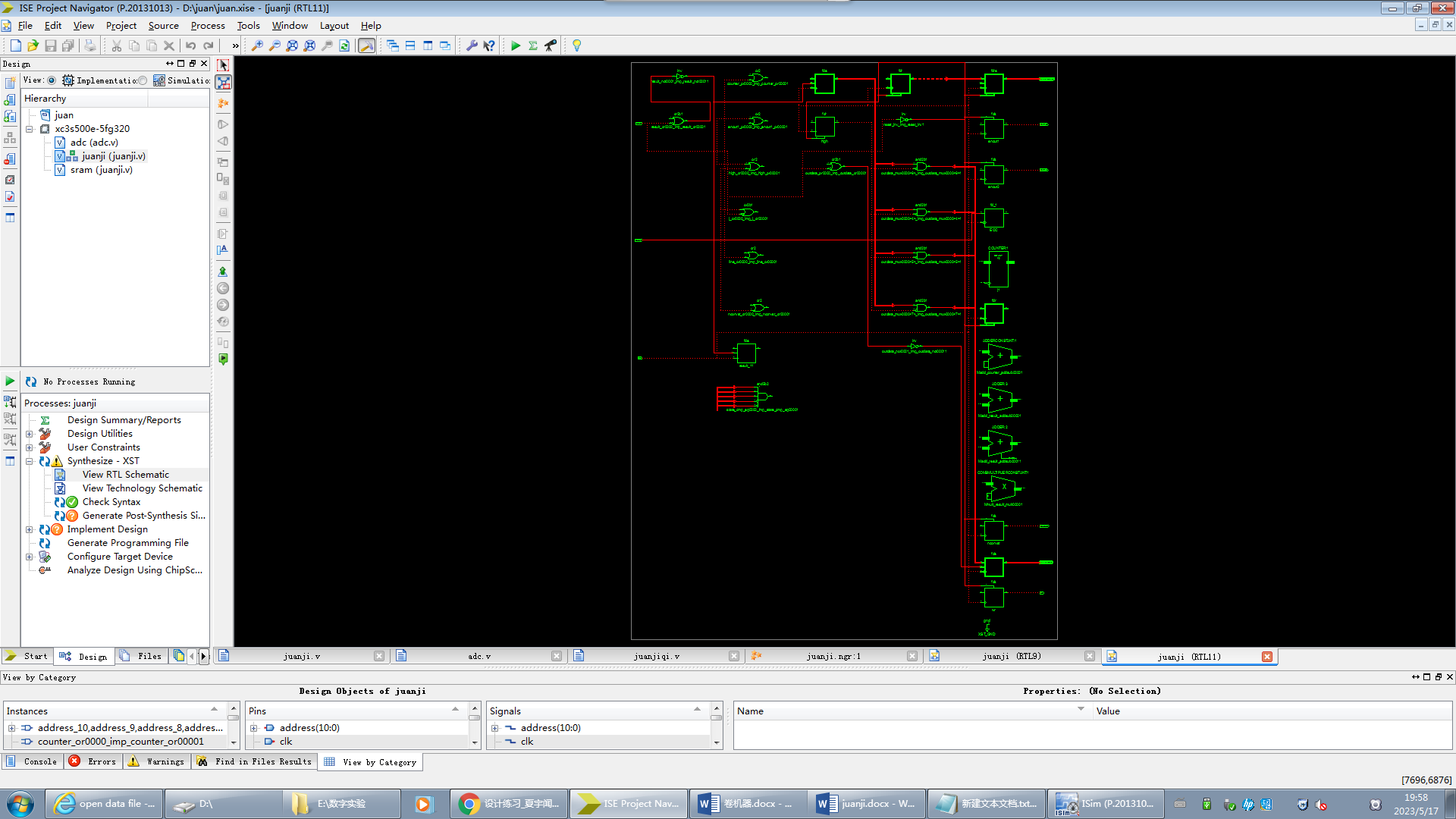
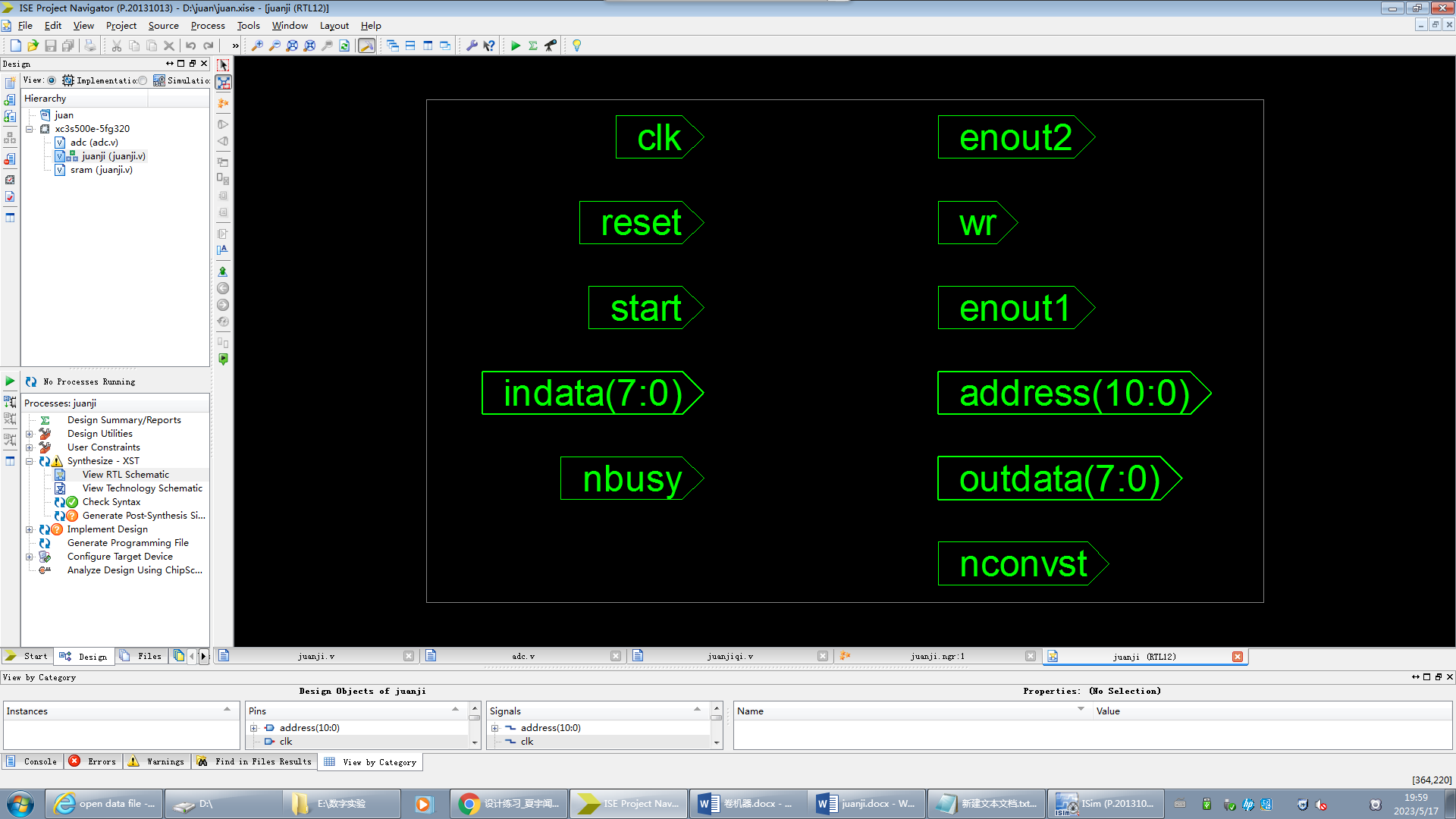
signals

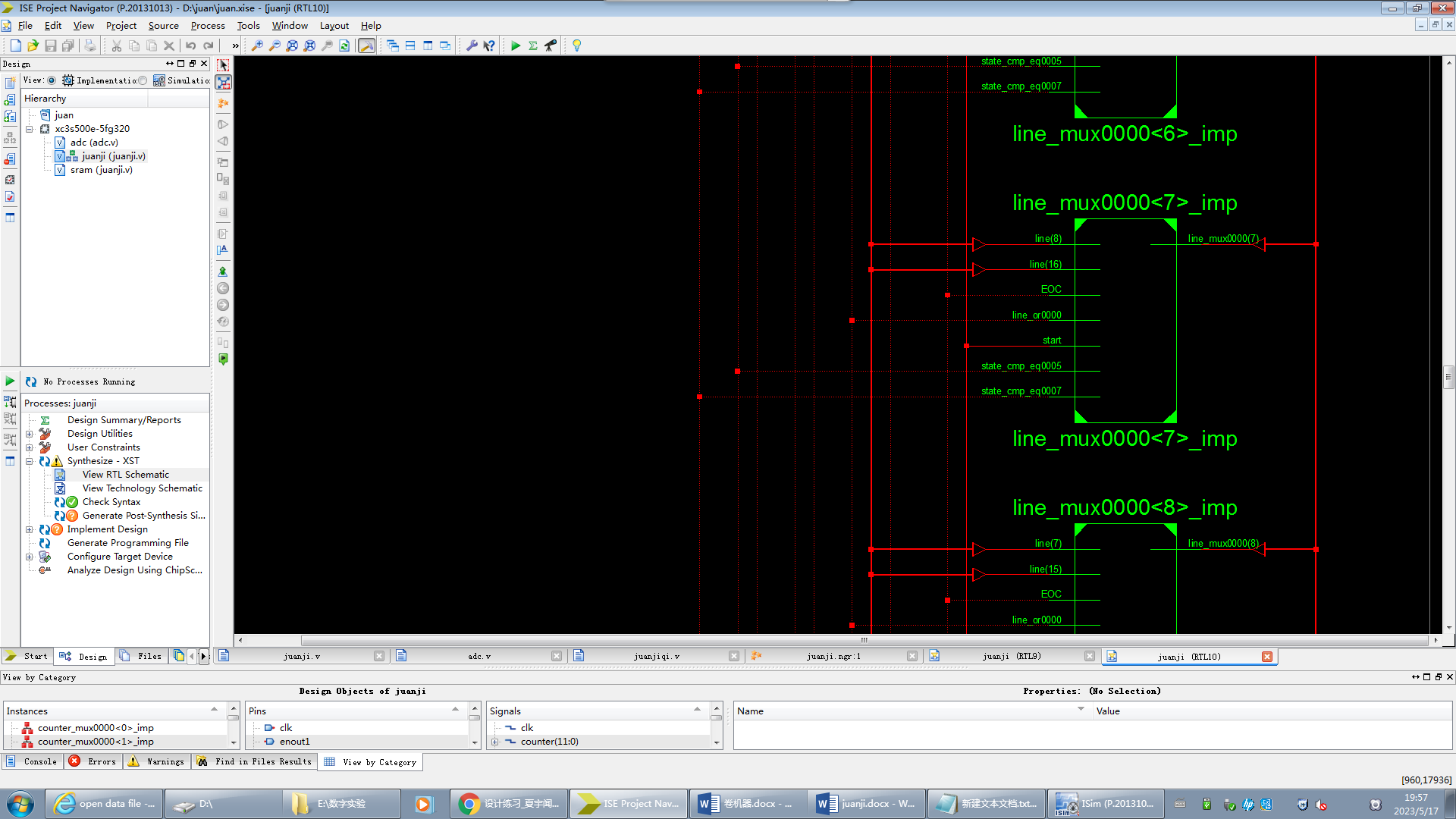


primitives

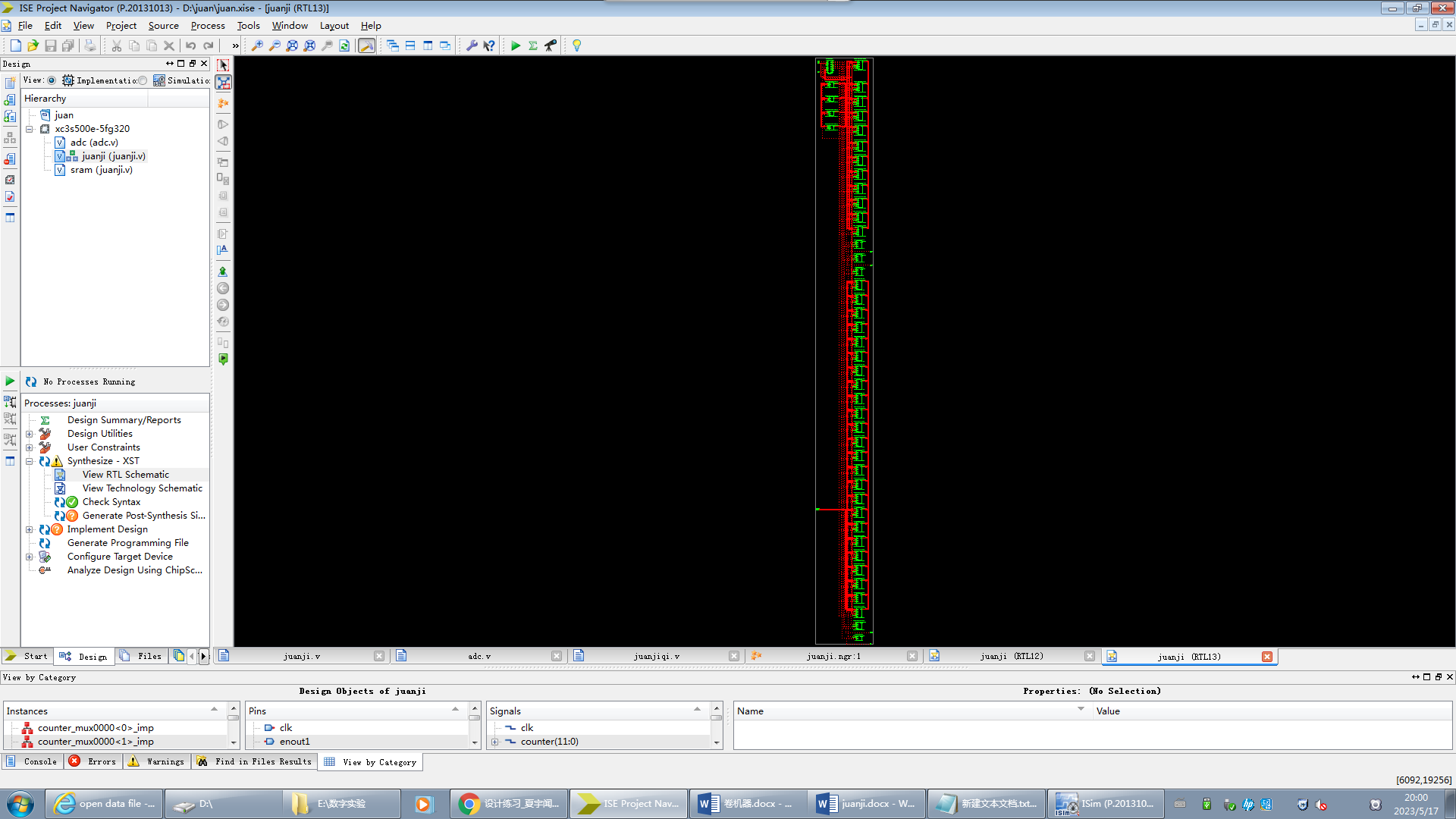


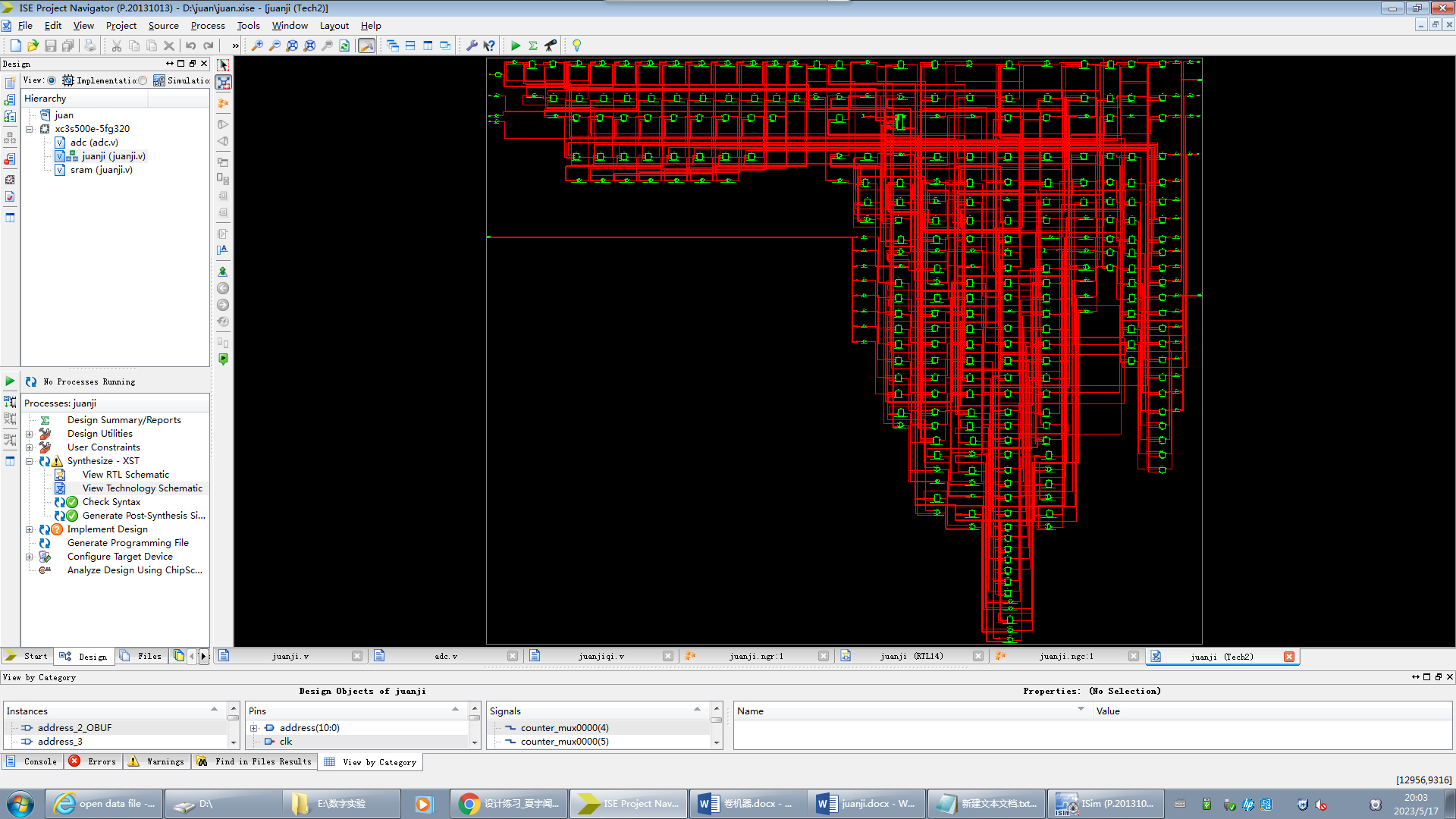
Top level ports

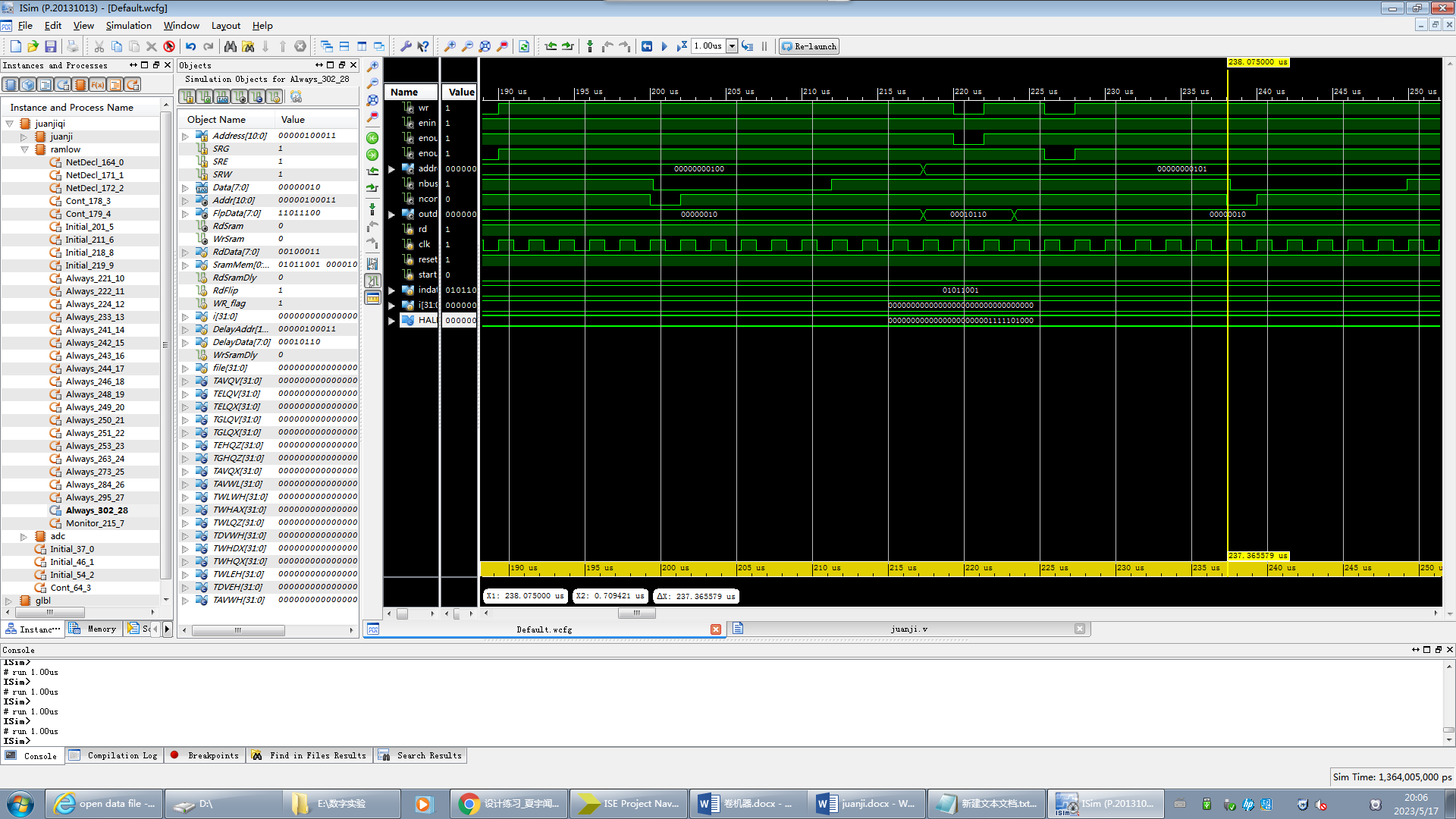


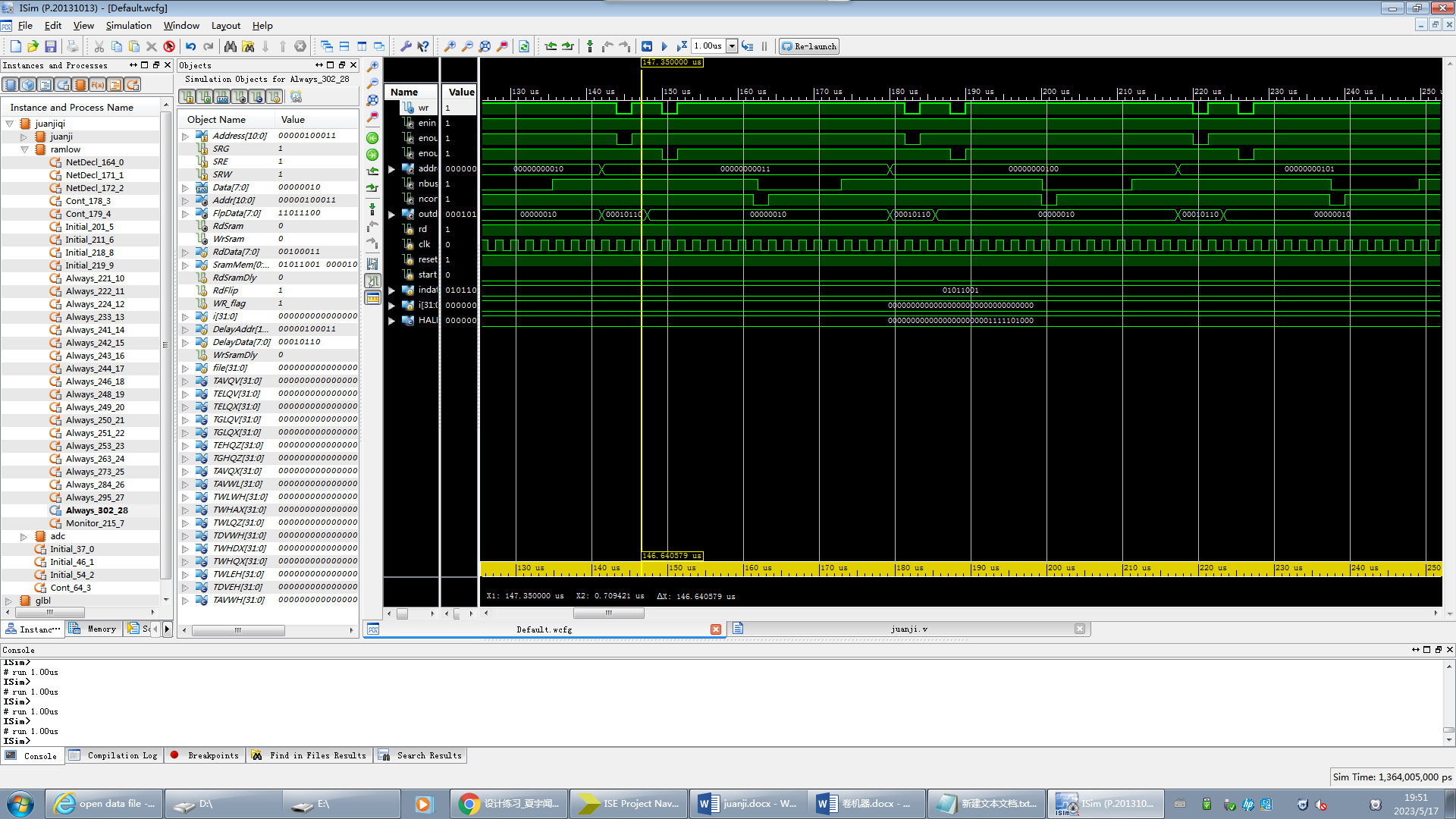


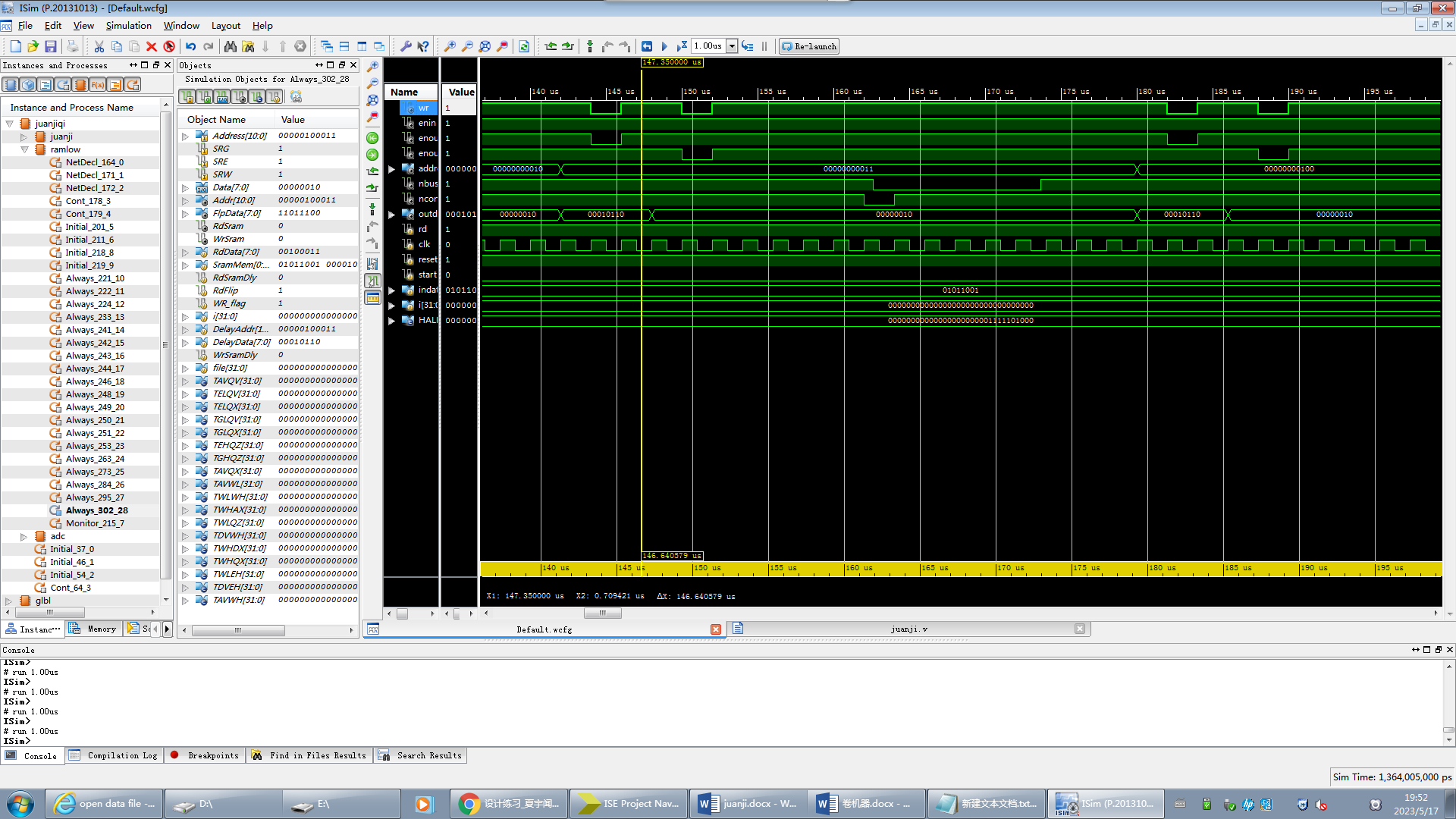
Jmp











module juanji(address,indata,outdata,wr,nconvst,nbusy,enout1,enout2,clk,reset,start);

input clk, //采用10MHz的时钟

reset, //复位信号

start, //因为RAM的空间是有限的，当RAM存满后采样和卷积都会停止，此时给一个start的高电平脉冲将会开始下一次的卷积

nbusy; //从A/D转换器来的信号表示转换器的忙或闲

output wr, //RAM写控制信号

enout1,enout2, //enout1是存储卷积低字节结果RAM的片选信号

//enout2是存储卷积高字节结果RAM的片选信号

nconvst, //给出A/D转换器的控制信号，命令转换器开始工作，低电平有效

address; //地址输出

input[7:0] indata; //从A/D转换器来的数据总线

output[7:0] outdata; //写到RAM去的数据总线

wire nbusy;

reg wr;

reg nconvst,enout1,enout2;

reg[7:0] outdata;

reg[10:0] address;

reg[8:0] state;

reg[15:0] result;

reg[23:0] line;

reg[11:0] counter;

reg high;

reg[4:0] j;

reg EOC;

parameter h1=1,h2=2,h3=3; //假设的系统系数

parameter IDLE=9'b000000001,START=9'b000000010,NCONVST=9'b000000100,

READ=9'b000001000,CALCU=9'b000010000,WRREADY=9'b000100000,

WR =9'b001000000,WREND=9'b010000000,WAITFOR=9'b100000000;

parameter FMAX=20; //因为A/D转换的时间是随机的，为保证按一定的频率采样，

//A/D转换控制信号应以一定频率给出。这里的采样频率可通过FMAX控制，

//并设为500kHz

always@(posedge clk)

if(!reset)

begin

state <= IDLE;

nconvst <= 1'b1;

enout1 <= 1;

enout2 <= 1;

counter <= 12'b0;

high <= 0;

wr <= 1;

line <= 24'b0;

address <= 11'b0;

end

else

case(state)

IDLE:

if(start==1)

begin

counter <=0; //counter是一个计数器，记录已用的RAM空间

line <=24'b0;

state <=START;

end

else

state <=IDLE;

START: //START状态控制A/D开始转换

if(EOC)

begin

nconvst <=0;

high <=0;

state <=NCONVST;

end

else

state <=START;

NCONVST: //NCONVST状态是A/D转换保持阶段

begin

nconvst <=1;

state <=READ;

end

READ: //READ状态读取A/D转换结果，计算卷积结果

begin

if(EOC)

begin

line <={line[15:0],indata};

state <=CALCU;

end

else

state <=READ;

end

CALCU:

begin

result <=line[7:0]\*h1+line[15:8]\*h2+line[23:16]\*h3;

state <=WRREADY;

end

//将卷积结果写入RAM时，先写入低字节，再写入高字节

WRREADY: //WRREADY状态是写RAM准备状态，建立地址和数据信号

begin

address <=counter;

if(!high)

outdata <=result[7:0];

else

outdata <=result[15:8];

state <=WR;

end

WR: //WR状态产生片选和写脉冲

begin

if(!high)

enout1 <=0;

else

enout2 <=0;

wr <=0;

state <=WREND;

end

WREND: //WREND状态结束一次写操作,若还未写入高字节则转到WRREADY状态开始高字节写入

begin

wr <=1;

enout1 <=1;

enout2 <=1;

if(!high)

begin

high <=1;

state <=WRREADY;

end

else

state <=WAITFOR;

end

WAITFOR: //WAITFOR状态控制采样频率并判断RAM是否已被写满

begin

if(j==FMAX-1)

begin

counter<=counter+1;

if(!counter[11])

state<=START;

else

begin

state<=IDLE;

$display($time,"The ram is used up.");

$stop;

end

end

else

state <=WAITFOR;

end

default: state <= IDLE;

endcase

assign rd = 1; //RAM的读信号始终保持高电平

//记录时钟，与FMAX共同控制采样频率，由于直接用clk的上升沿对nbusy判断，以决定某些操作是否运行时，

//会因为两个信号的跳变沿相隔太近而令状态机不能正常工作，因此利用clk的下降沿建立EOC信号与nbusy同步，

//相位相差180°，然后用clk的上升沿判断操作是否进行

always@(negedge clk)

begin

EOC <= nbusy;

if(!reset||state==START)

j<=1;

else

j<=j+1;

end

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

module adc(nconvst,nbusy,data);

input nconvst; //A/D启动脉冲ST

output nbusy; //A/D工作标志

output data; //数据总线，从AD.DATA文件中读取数据后经端口输出

reg[7:0] databuf,i; //内部寄存器

reg nbusy;

wire[7:0] data;

reg[7:0] data\_mem[0:255];

reg link\_bus;

integer tconv,t5,t8,t9,t12;

integer wideth1,wideth2,wideth;

//时间参数定义

always@(negedge nconvst)

begin

tconv=9500+{$random}%500; //(type 950,max 1000ns)Conversion Time

t5={$random}%1000; //(max 100ns) CONVST to BUSY Propagation Dlay

//CL=10pf

t8=200; //(min 20ns) CL=20pF Data Setup Time Prior to BUSY\

//(min 10ns) CL=100pF

t9=100+{$random}%900; //(min 10ns,max 100ns) Bus Relinquish Time After CONVST

t12=2500; //(type) BUSY High to CONVST Low, SHA Aquistion Time

end

initial

begin

$readmemh("adc.data",data\_mem); //从数据文件adc.data中读取数据

i=0;

nbusy=1;

link\_bus=0;

end

assign data = link\_bus?databuf:8'bzz; //三态总线

always@(negedge nconvst)

fork

#t5 nbusy=0;

@(posedge nconvst)

begin

#tconv nbusy=1;

end

join

always@(negedge nconvst)

begin

@(posedge nconvst)

begin

#(tconv-t8) databuf=data\_mem[i];

end

if(wideth<10000&&wideth>500)

begin

if(i==255)

i=0;

else

i=i+1;

end

else

i=i;

end

//在模数转换期间关闭三态输出，转换结束时启动三态输出

always@(negedge nconvst)

fork

#t9 link\_bus=1'b0; //关闭三态输出，不允许总线输出

@(posedge nconvst)

begin

#(tconv-t8) link\_bus=1'b1;

end

join

//检查A/D启动信号的频率是否太快

always@(posedge nbusy)

begin

#t12;

if(!nconvst)

begin

$display("Warning! SHA Acquisition Time is too short!");

end

else

$display("Warning! SHA Acquisition Time is enough!");

end

//坚持A/D启动信号的负脉冲宽度是否足够和太宽

always@(negedge nconvst)

begin

wideth=$time;

@(posedge nconvst) wideth=$time-wideth;

if(wideth<=500||wideth>10000)

begin

$display("nCONVST Pulse Wideth=%d",wideth);

$display("Warning! nCONVST Pulse Width is too narrow or too wide!");

$stop;

end

end

endmodule

module juanji(address,indata,outdata,wr,nconvst,nbusy,enout1,enout2,clk,reset,start);

input clk, //采用10MHz的时钟

reset, //复位信号

start, //因为RAM的空间是有限的，当RAM存满后采样和卷积都会停止，此时给一个start的高电平脉冲将会开始下一次的卷积

nbusy; //从A/D转换器来的信号表示转换器的忙或闲

output wr, //RAM写控制信号

enout1,enout2, //enout1是存储卷积低字节结果RAM的片选信号

//enout2是存储卷积高字节结果RAM的片选信号

nconvst, //给出A/D转换器的控制信号，命令转换器开始工作，低电平有效

address; //地址输出

input[7:0] indata; //从A/D转换器来的数据总线

output[7:0] outdata; //写到RAM去的数据总线

wire nbusy;

reg wr;

reg nconvst,enout1,enout2;

reg[7:0] outdata;

reg[10:0] address;

reg[8:0] state;

reg[15:0] result;

reg[23:0] line;

reg[11:0] counter;

reg high;

reg[4:0] j;

reg EOC;

parameter h1=1,h2=2,h3=3; //假设的系统系数

parameter IDLE=9'b000000001,START=9'b000000010,NCONVST=9'b000000100,

READ=9'b000001000,CALCU=9'b000010000,WRREADY=9'b000100000,

WR =9'b001000000,WREND=9'b010000000,WAITFOR=9'b100000000;

parameter FMAX=20; //因为A/D转换的时间是随机的，为保证按一定的频率采样，

//A/D转换控制信号应以一定频率给出。这里的采样频率可通过FMAX控制，

//并设为500kHz

always@(posedge clk)

if(!reset)

begin

state <= IDLE;

nconvst <= 1'b1;

enout1 <= 1;

enout2 <= 1;

counter <= 12'b0;

high <= 0;

wr <= 1;

line <= 24'b0;

address <= 11'b0;

end

else

case(state)

IDLE:

if(start==1)

begin

counter <=0; //counter是一个计数器，记录已用的RAM空间

line <=24'b0;

state <=START;

end

else

state <=IDLE;

START: //START状态控制A/D开始转换

if(EOC)

begin

nconvst <=0;

high <=0;

state <=NCONVST;

end

else

state <=START;

NCONVST: //NCONVST状态是A/D转换保持阶段

begin

nconvst <=1;

state <=READ;

end

READ: //READ状态读取A/D转换结果，计算卷积结果

begin

if(EOC)

begin

line <={line[15:0],indata};

state <=CALCU;

end

else

state <=READ;

end

CALCU:

begin

result <=line[7:0]\*h1+line[15:8]\*h2+line[23:16]\*h3;

state <=WRREADY;

end

//将卷积结果写入RAM时，先写入低字节，再写入高字节

WRREADY: //WRREADY状态是写RAM准备状态，建立地址和数据信号

begin

address <=counter;

if(!high)

outdata <=result[7:0];

else

outdata <=result[15:8];

state <=WR;

end

WR: //WR状态产生片选和写脉冲

begin

if(!high)

enout1 <=0;

else

enout2 <=0;

wr <=0;

state <=WREND;

end

WREND: //WREND状态结束一次写操作,若还未写入高字节则转到WRREADY状态开始高字节写入

begin

wr <=1;

enout1 <=1;

enout2 <=1;

if(!high)

begin

high <=1;

state <=WRREADY;

end

else

state <=WAITFOR;

end

WAITFOR: //WAITFOR状态控制采样频率并判断RAM是否已被写满

begin

if(j==FMAX-1)

begin

counter<=counter+1;

if(!counter[11])

state<=START;

else

begin

state<=IDLE;

$display($time,"The ram is used up.");

$stop;

end

end

else

state <=WAITFOR;

end

default: state <= IDLE;

endcase

assign rd = 1; //RAM的读信号始终保持高电平

//记录时钟，与FMAX共同控制采样频率，由于直接用clk的上升沿对nbusy判断，以决定某些操作是否运行时，

//会因为两个信号的跳变沿相隔太近而令状态机不能正常工作，因此利用clk的下降沿建立EOC信号与nbusy同步，

//相位相差180°，然后用clk的上升沿判断操作是否进行

always@(negedge clk)

begin

EOC <= nbusy;

if(!reset||state==START)

j<=1;

else

j<=j+1;

end

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 / 100ps

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

// Company:

// Engineer:

//

// Create Date: 18:56:49 05/17/2023

// Design Name: juanji

// Module Name: D:/juan/juanjiqi.v

// Project Name: juan

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: juanji

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module juanjiqi;

wire wr,enin,enout1,enout2;

wire[10:0] address;

reg rd,clk,reset,start;

reg[7:0] indata;

wire nbusy,nconvst;

wire[7:0] outdata;

integer i;

parameter HALF\_PERIOD=1000;

//产生10kHz的时钟

initial

begin

rd=1;

i=0;

clk=1;

forever #HALF\_PERIOD clk = ~clk;

end

//产生置位信号

initial

begin

reset=1;

#(HALF\_PERIOD\*2+50) reset=0;

#(HALF\_PERIOD\*3) reset=1;

end

//产生开始卷积控制信号

initial

begin

start=0;

indata=8'b01011001;

#(HALF\_PERIOD\*7+20) start=1;

#(HALF\_PERIOD\*2) start=0;

#(HALF\_PERIOD\*1000) start=1;

#(HALF\_PERIOD\*2) start=0;

end

assign enin = 1;

juanji juanji(

.address(address),

.clk(clk),

.enout1(enout1),

.enout2(enout2),

.indata(indata),

.nbusy(nbusy),

.nconvst(nconvst),

.outdata(outdata),

.reset(reset),

.start(start),

.wr(wr)

);

sram ramlow(.Address(address),.Data(outdata),.SRW(wr),.SRG(rd),.SRE(enout1));

adc adc(.nconvst(nconvst),.nbusy(nbusy),.data(indata));

endmodule