**MIPS单周期微处理器设计**

**通信2002班 涂增基 U202013990**

1. **实验任务**

利用HDL语言，基于Xilinx FPGA nexys4实验平台，设计一个能够执行以下MIPS指令集的单周期类MIPS处理器，要求完成所有支持指令的功能仿真，验证指令执行的正确性

支持基本的算术逻辑运算如add，sub，and，or，slt指令

支持基本的内存操作如lw，sw指令

支持基本的程序控制如beq，j指令

仿真的汇编程序代码见超星慕课，3.5节 实验任务

1. **实验目标**

1.了解微处理器的基本结构

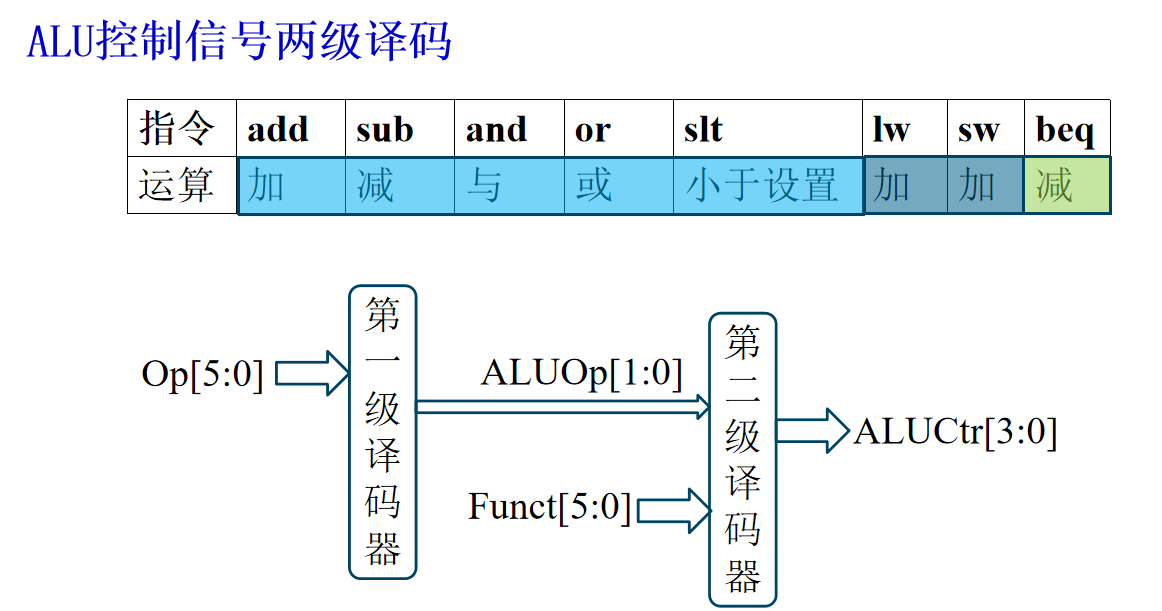
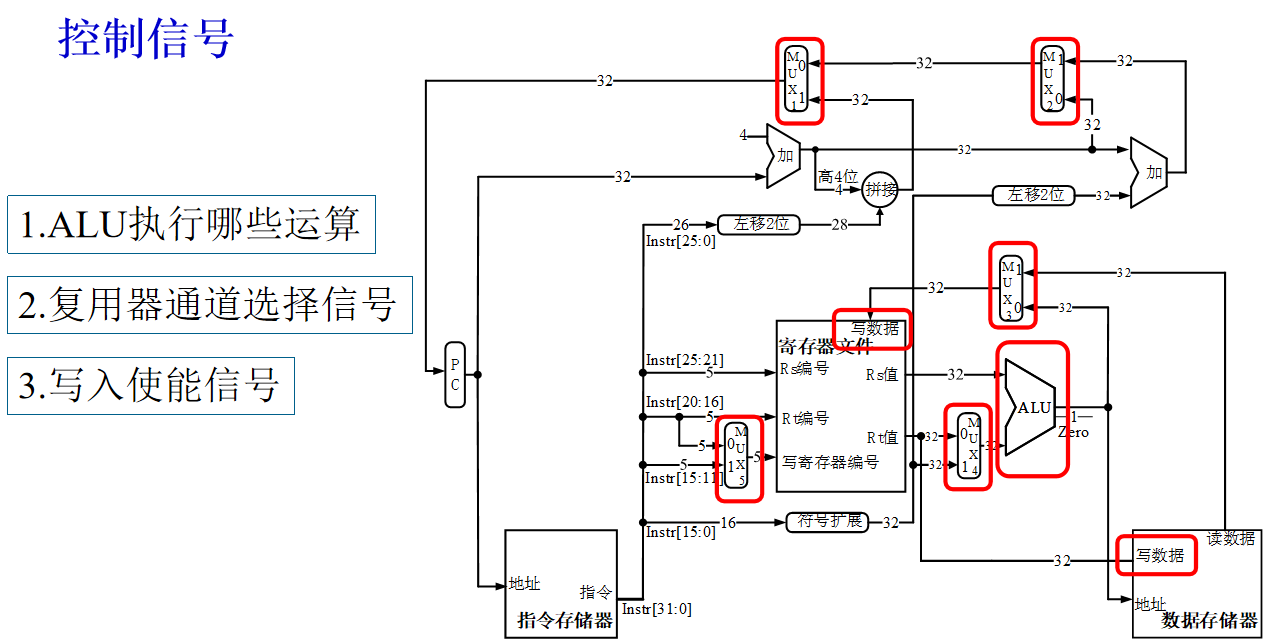
2.掌握哈佛结构的计算机工作原理

3.学会设计简单的微处理器

4.了解软件控制硬件工作的基本原理

1. **程序设计思路**

简单指令集MIPS微处理器完整数据通路和加上控制器后的通路图如下，先搭建出每个独立模块，再组合成整体控制器。自顶向下。



1. **源代码及注释**

**1、寄存器PC**

`timescale 1ns / 1ps

module PC(D, Clk, Reset, Q);//寄存器 PC，

parameter n=32;

input [n-1:0] D;

input Clk, Reset;

output reg [n-1:0] Q; //复位时 Q 为 0.

always @(posedge Clk or posedge Reset)

begin //时钟 Clk 上升沿 Q 锁存输入数据 D, 异步复位信号 Reset 高电平有效

if(Reset)

Q<=0;

else

Q<=D;

end

endmodule

**2、指令存储器InstrROM**

`timescale 1ns / 1ps

module InstrROM(Addr,Clk,Instr); //输出指令存储器 InstrROM，容量为 2n\*32

parameter n=5;

input [n-1:0] Addr; //输入引脚：地址线 Addr (位宽 n)、同步时钟 Clk

input Clk;

output reg [31:0] Instr; //输出引脚：指令 Instr（位宽 32，具有保持功能）

reg [31:0] regs[2\*\*n-1:0];

always @ (posedge Clk)

Instr = regs[Addr]; //时钟上升沿输出相应地址的指令

endmodule

**3、控制器 Controller**

`timescale 1ns / 1ps

module Controller (OpCode, Funct, J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUCtr); //控制器

input [5:0] OpCode, Funct;

output J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;

output [3:0] ALUCtr;

wire [1:0] ALUOp;

//引用主控制器 Mainctr 以及 ALU 控制器 ALUControl 实现对指令的操作码和功能码译码产生控制信号

MainCtr U0(OpCode,J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp);//主控制器第一级译码,得到ALUOp[1:0]

ALUControl U1(ALUOp, Funct, ALUCtr); //ALU控制器第二级译码，产生ALUCtr[3:0]

endmodule

**3.1 主控制器 Mainctr**

`timescale 1ns / 1ps

module MainCtr(OpCode,J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp);//主控制器 Mainctr

input [5:0] OpCode;

output J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;

output [1:0] ALUOp;

reg [8:0] temp;

assign{J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp}=temp;

always @(OpCode) //对6位操作码Op[5:0]进行译码

case(OpCode)

6'b000000:temp=9'b001100010;

6'b100011:temp=9'b000110100;

6'b101011:temp=9'b00x011x00;

6'b000100:temp=9'b01x000x01;

6'b000010:temp=9'b1xx0x0xxx;

default:temp=9'b000000000;

endcase

endmodule

**3.2 ALU 控制器 ALUControl**

`timescale 1ns / 1ps

module ALUControl (ALUOp, Funct, ALUCtr);// ALU 控制器 ALUControl

input [1:0] ALUOp;

input [5:0] Funct;

output reg [3:0] ALUCtr;

always @(\*) //对功能码Funct[5:0]和ALUOp[1：0]进行译码

casex({ALUOp, Funct})

8'b00xxxxxx:ALUCtr=4'b0010;

8'b00xxxxxx:ALUCtr=4'b0010;

8'b01xxxxxx:ALUCtr=4'b0110;

8'b10100000:ALUCtr=4'b0010;

8'b10100010:ALUCtr=4'b0110;

8'b10100100:ALUCtr=4'b0000;

8'b10100101:ALUCtr=4'b0001;

8'b10101010:ALUCtr=4'b0111;

endcase

endmodule

**4、多路复用器 Mux2\_1**

`timescale 1ns / 1ps

module Mux2\_1 (In1, In2, sel, Out); //多路复用器 Mux2\_1

parameter n=32;

input [n-1:0] In1, In2; //输入：In1（n 位），In2（n 位），sel

input sel;

output [n-1:0] Out; //输出：Out（n 位）

//将两个 n 位宽的输入通道复用到一个输出同位宽通道，当 sel 为 0 时，选择 In1 输出，否则选择 In2 输出

assign Out = sel?In2:In1;

endmodule

**5、3 选 1 多路复用器 Mux3\_1**

`timescale 1ns / 1ps

module Mux3\_1 (In1, In2,In3, sel, Out);

//n 位 3 选 1 多路复用器 Mux3\_1 基于 n 位 2 选 1 多路复用器构建

parameter n=32;

input [n-1:0] In1, In2,In3;

input [1:0] sel;

output [n-1:0] Out;

wire [n-1:0] OutTemp;

Mux2\_1 U0(In1, In2, sel[0], OutTemp);

Mux2\_1 U1(OutTemp, In3, sel[1], Out);

endmodule

**6、左移运算器 LeftShift**

`timescale 1ns / 1ps

module LeftShift (In, Out);

//左移运算器 LeftShift 将 n 位数据左移 x 位(低位补充 0，高位移出)输出 m 位数据

parameter n=32,m=32,x=2;

input [n-1:0] In;

output [m-1:0] Out;

assign Out =In << x;

endmodule

**7、加法器 Adder**

`timescale 1ns / 1ps

module Adder (In1,In2,Out);

//加法器 Adder 将两个 n 位宽的数据相加，输出结果

parameter n=32;

input [n-1:0] In1,In2;

output [n-1:0] Out;

assign Out =In1 +In2;

endmodule

**8、寄存器文件 RegFile**

`timescale 1ns / 1ps

module RegFile (RsAddr, RtAddr, WrAddr, DataIn, RegWr,Clk, RsData, RtData);

//同步输入\异步输出寄存器文件 RegFile，容量为 32\*32，且编号为 0 的寄存器取值恒为 0

//输入引脚：Rs 寄存器编号 RsAddr (位宽 5)、Rt 寄存器编号 RtAddr (位宽 5)，写寄存器编号 WrAddr（(位宽 5)）输入数据线 DataIn（位宽

//32），写控制信号 RegWr（高电平有效），同步时钟 Clk

input [4:0] RsAddr,RtAddr, WrAddr;

input [31:0] DataIn;

input RegWr,Clk;

//输出引脚：Rs 寄存器数据 RsData（位宽 32），Rt 寄存器数据 RtData（位宽 32）

output [31:0] RsData, RtData;

reg [31:0] regs[31:0];

//R型指令执行时，首先根据指令字段Instr[25...21],Instr[20...16]获取$Rs,$Rt的值

assign RsData=RsAddr?regs[RsAddr]:0;

assign RtData=RtAddr?regs[RtAddr]:0;

always @(posedge Clk)

if(RegWr)

regs[WrAddr]=DataIn;

endmodule

**9.符号扩展模块 SignedExtend**

`timescale 1ns / 1ps

module SignedExtend (In, Out);

//将 n 位符号数扩展为 m 位符号数模块 SignedExtend

parameter n=16,m=32;

input [n-1:0] In;

output reg [m-1:0] Out;

integer i;

always @(In)

begin

**if(In[n-1])**  //有符号数则最高位补1

for(i=n;i<m;i=i+1)

Out[i]=1'b1;

else

//无符号数则最高位补0

for(i=n;i<m;i=i+1)

Out[i]=1'b0;

Out[n-1:0]=In;

end

endmodule

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

//另一种实现方案

//module SignedExtend #(

// parameter n = 16,

// m = 32

//)(

// input [n-1:0] In,

// output [m-1:0] Out

// );

//

// **assign Out = {{(m-n){In[n-1]}},In};**

//endmodule

**10、32 位宽算术逻辑运算单元 ALU**

module ALU( //32 位宽算术逻辑运算单元 ALU

input signed [31:0] In1,

input signed [31:0] In2,

input [3:0] ALUCtr,

//输入引脚：输入数据源 1-In1（32 位），输入数据源 2-In2（32 位），运算类型控制-ALUCtr（3 位）

output reg [31:0] ALURes,

output Zero

//输出引脚：运算结果 Res（32 位宽）

);

//运算结果零标志 Zero（1 位，如果 Res 不为零 Zero 为 0，否则 Zero 为 1

assign Zero=(ALURes==0)?1:0;

always @(In1 or In2 or ALUCtr)

begin

case(ALUCtr)

4'b0110://sub

begin

ALURes = In1-In2;

end

4'b0010://add

begin

ALURes = In1+In2;

end

4'b0000://and

begin

ALURes = In1 & In2;

end

4'b0001://or

begin

ALURes = In1 | In2;

end

4'b0111://slt

begin

ALURes = (In1<In2)?1:0;

end

default:

begin

ALURes=0;

end

endcase

end

endmodule

**11、位拼接器 Concat**

`timescale 1ns / 1ps

module Concat (In1, In2, Out);

//位拼接器 Concat 将输入源 In1（n 位），In2（m 位）拼接合成为 Out（n+m 位，且 In1 在高位部分，In2 在低位部分）

parameter n=4,m=28;

input [n-1:0] In1;

input [m-1:0] In2;

output [n+m-1:0] Out;

assign Out = {In1,In2};

endmodule

**12、数据存储器 DataRAM**

`timescale 1ns / 1ps

module DataRAM(Addr,DataIn,MemWR,Clk,DataOut);

//同步（时钟上升沿）输入\异步输出数据存储器 DataRAM，容量为 2n\*m

parameter n=5,m=32;

reg [m-1:0] regs[2\*\*n-1:0];

input [n-1:0] Addr;

input [m-1:0] DataIn;

input MemWR,Clk;

output [m-1:0] DataOut;

//地址线 Addr (位宽 n)、输入数据线 DataIn（位宽 m），写控制信号 MemWR(高电平有效)，同步时钟 Clk

//数据输出 DataOut（位宽 m，具有保持功能）

assign DataOut = regs[Addr];

always @ (posedge Clk)

begin

if(MemWR)

regs[Addr] <= DataIn;

end

endmodule

**13、整体模块实现MIPSCPU 简单指令集MIPS微处理器**

`timescale 1ns /1ps

module MIPSCPU ( //简单指令集MIPS微处理器

input Clk,

input Reset

);

wire [31:0] CurrentPC,SequencePC,JumpPC,BranchPC,TempPC,Boffset,Instr,MemData,Data2Reg,Res,RsData,RtData,ALUIn2,SignExtended;

wire [1:0] ALUOp,PCsel;

wire [4:0] WrAddr;

wire J,B,RegDst,ALUSrc,RegWr,MemWr,Mem2Reg,Zero,BZero;

wire [27:0] JumpTarget;

wire [3:0] ALUCtr;

PC U0\_PC(TempPC, Clk, Reset, CurrentPC);

InstrROM ROM\_U0(CurrentPC[6:2],~Clk,Instr);

defparam ROM\_U0.n=5;

Controller U2\_Contr(Instr[31:26],Instr[5:0], J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUCtr);

Mux2\_1 U3(Instr[20:16], Instr[15:11],RegDst,WrAddr);

defparam U3.n=5;

LeftShift U4(Instr[25:0], JumpTarget);

defparam U4.n=26,U4.m=28,U4.x=2;

Adder U5(CurrentPC,'h4,SequencePC);

defparam U5.n=32;

Concat U6(SequencePC[31:28], JumpTarget, JumpPC);

defparam U6.n=4,U6.m=28;

Adder U7(SequencePC,Boffset,BranchPC);

defparam U7.n=32;

Mux3\_1 U8(SequencePC, BranchPC,JumpPC,PCsel, TempPC);

defparam U8.n=32;

LeftShift U9(SignExtended, Boffset);

defparam U9.n=32,U9.m=32,U9.x=2;

RegFile RegFile\_U1(Instr[25:21],Instr[20:16], WrAddr,Data2Reg, RegWr,Clk, RsData, RtData);

SignedExtend U11(Instr[15:0], SignExtended);

defparam U11.n=16,U11.m=32;

Mux2\_1 U12(RtData, SignExtended,ALUSrc,ALUIn2);

defparam U12.n=32;

ALU U13(RsData,ALUIn2,ALUCtr,Res,Zero);

Mux2\_1 U14(Res, MemData,Mem2Reg,Data2Reg);

defparam U14.n=32;

Concat U15(J, BZero, PCsel);

defparam U15.n=1,U15.m=1;

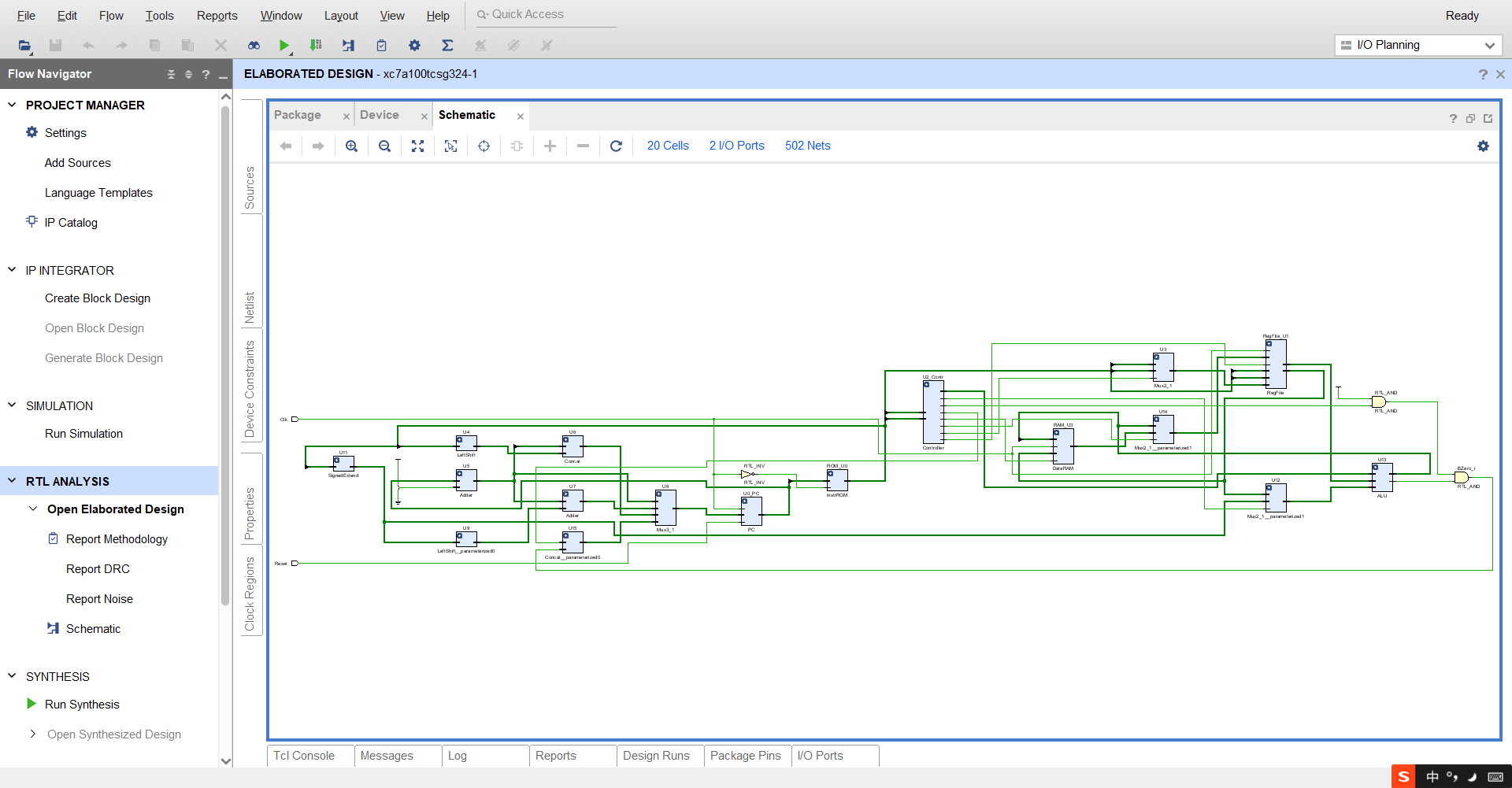
DataRAM RAM\_U3(Res[6:2],RtData,MemWr,Clk, MemData);

defparam RAM\_U3.n=5,RAM\_U3.m=32;

and(BZero,B,Zero);

endmodule

最终得到的整体电路图如下：



**五、仿真代码及波形**

**1、寄存器PC**

`timescale 1ns / 1ps

module PCsim( );

reg [31:0] D;

reg Clk, Reset;

wire [31:0] Q;

PC U0(D, Clk, Reset, Q);

defparam U0.n=32;

parameter PERIOD = 10;

always begin

Clk = 1'b0;

#(PERIOD/2) Clk = 1'b1;

#(PERIOD/2);

end

integer i;

initial

begin

Reset=0;

for(i=0;i<5;i=i+1)

begin

D=i;

#10;

end

#10;

Reset=1;

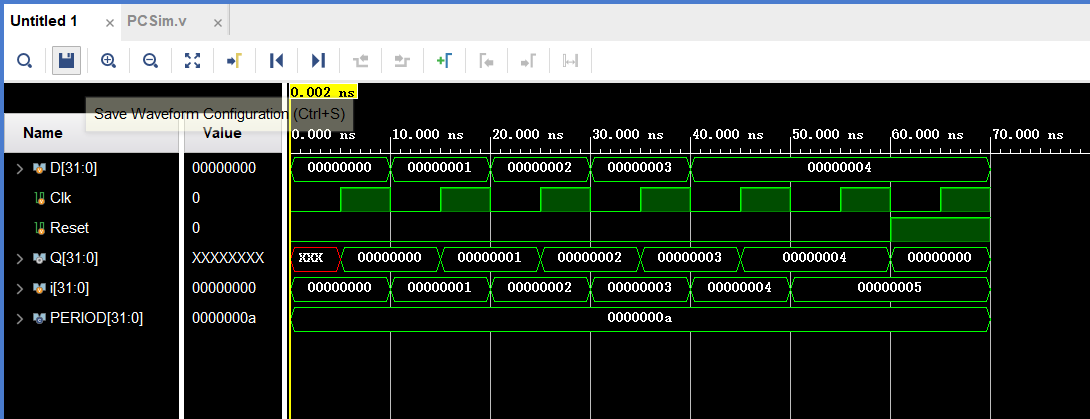
#10;

$stop;

end

endmodule

**仿真波形**



分析波形可知，在每一个时钟上升沿，Q的值变为与D一样，符合D触发器的功能

**2、指令存储器InstrROM**

`timescale 1ns / 1ps

module InstrROMsim();

reg [4:0] Addr;

reg Clk;

wire [31:0] Instr;

InstrROM U0\_ROM(Addr,Clk,Instr);

defparam U0\_ROM.n=5;

parameter PERIOD = 10;

always begin

Clk = 1'b0;

#(PERIOD/2) Clk = 1'b1;

#(PERIOD/2);

end

integer i;

initial

begin

**for(i=0;i<32;i=i+1)**

**U0\_ROM.regs[i]=i\*4;**

**for(i=0;i<32;i=i+1)**

**begin**

**Addr=i;**

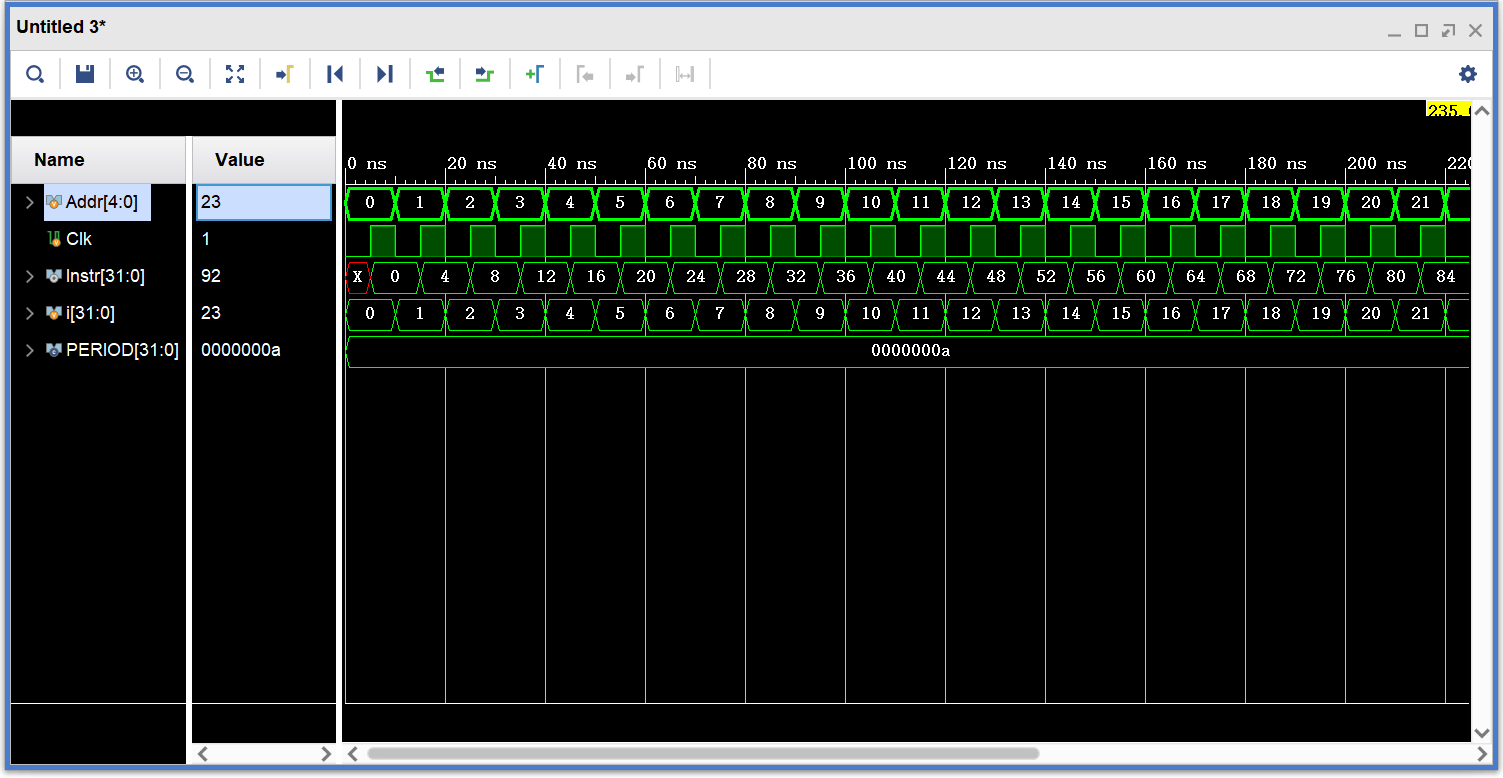
#10;

end

$stop;

end

endmodule



仿真中实例化 32 个存储单元，每个存储单元的初始值为其地址\*4 ，在时钟上升沿输出每个地址的指令，如图可知符合要求。

1. **控制器Controller**

`timescale 1ns / 1ps

module Controllersim( );

reg [5:0] OpCode, Funct;

wire J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;

wire [3:0] ALUCtr;

Controller UO\_Controller(OpCode, Funct, J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUCtr);

initial

begin

OpCode=6'h00;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

#10;

end

OpCode=6'h23;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

#10;

end

OpCode=6'h2b;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

#10;

end

OpCode=6'h04;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

#10;

end

OpCode=6'h02;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

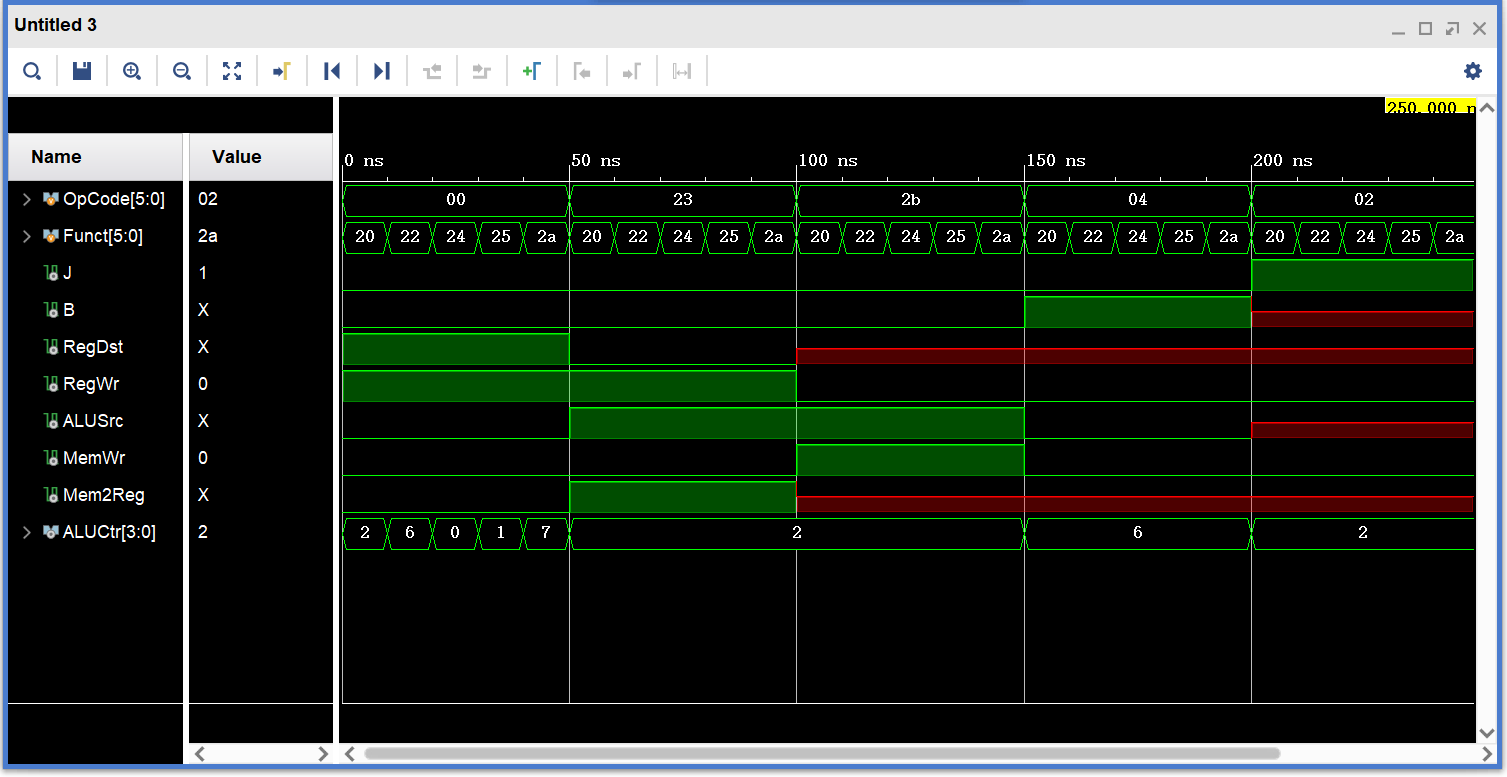
#10;

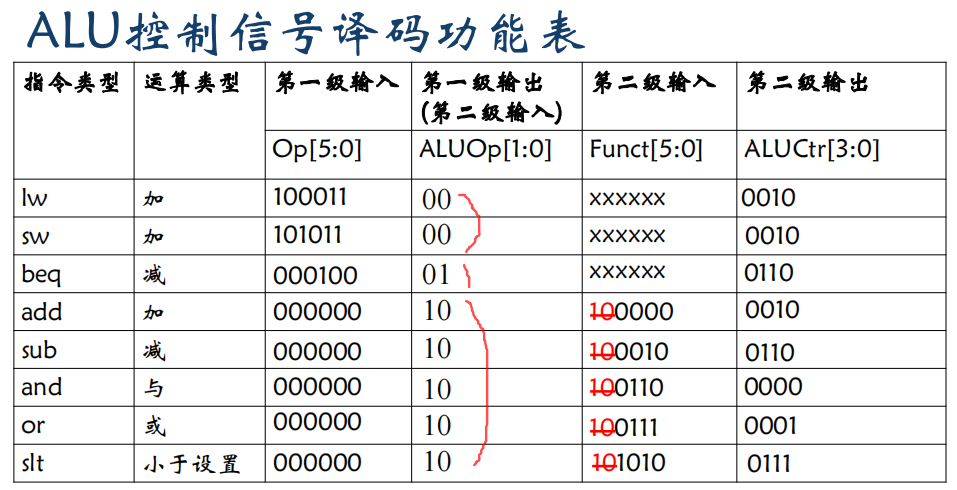
end

$stop;

end

endmodule





**由波形图和真值表可知，**R型 Op：000000 ，I型无Funct[5:0],由Op直接决定运算类型，当Op为00，Funct分别为20,22,24，25,2a时,ALUCtr分别为2,6,0，1，7，对应加，减与或小余运算，其他同理，符合要求。

**3.1 MainCtrl**

`timescale 1ns / 1ps

module MainCtrlsim( );

reg [5:0] OpCode;

wire J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg;

wire [1:0] ALUOp;

MainCtr UO\_MainCtr(OpCode,J, B, RegDst, RegWr, ALUSrc, MemWr, Mem2Reg, ALUOp);

initial

begin

OpCode =6'h00;

#10;

OpCode = 6'h23;

#10;

OpCode = 6'h2b;

#10;

OpCode =6'h04;

#10;

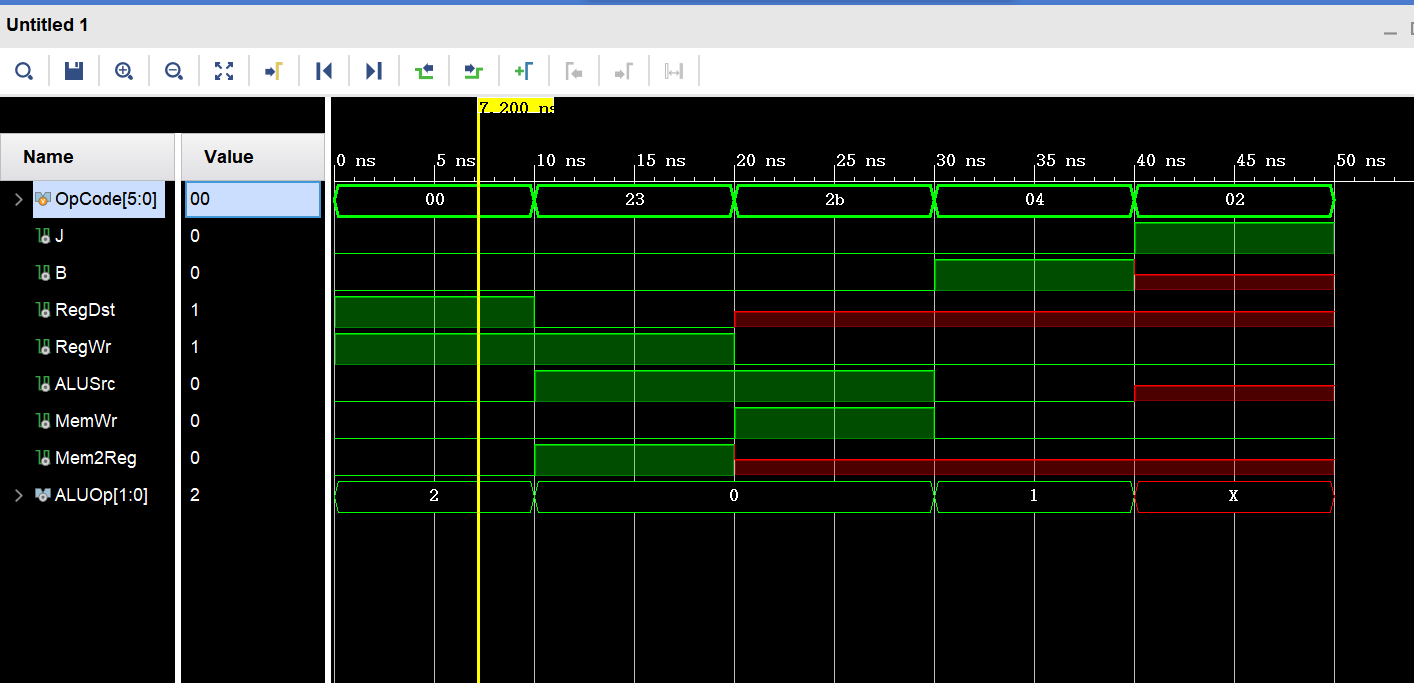
OpCode =6'h02;

#10;

$stop;

end

endmodule



**3.2 ALUCtrsim**

`timescale 1ns / 1ps

module ALUControlsim();

reg [1:0] ALUOp;

reg [5:0] Funct;

wire [3:0] ALUCtr;

ALUControl U0\_ALUControl(ALUOp, Funct, ALUCtr);

integer i;

initial

begin

for (i=0;i<3;i=i+1)

begin

ALUOp=i;

begin

Funct=6'h20;

#10;

Funct=6'h22;

#10;

Funct=6'h24;

#10;

Funct=6'h25;

#10;

Funct=6'h2a;

#10;

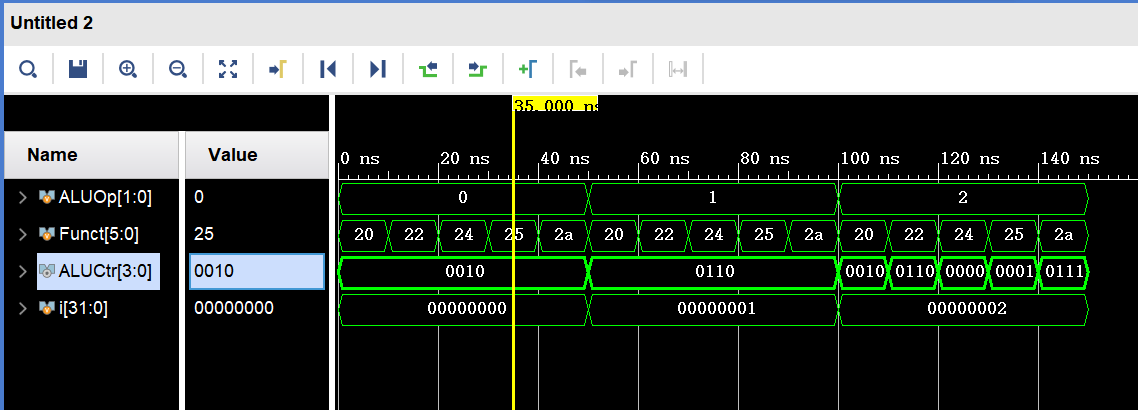
end

end

$stop;

end

endmodule



1. **多路复用器 Mux2\_1**

`timescale 1ns / 1ps

module Muxsim( );

reg [31:0] In1, In2;

reg sel;

wire [31:0] Out;

Mux2\_1 U0(In1, In2, sel, Out);

defparam U0.n=32;

initial

begin

In1=32'h12345678;

In2=32'h87654321;

sel=0;

#10;

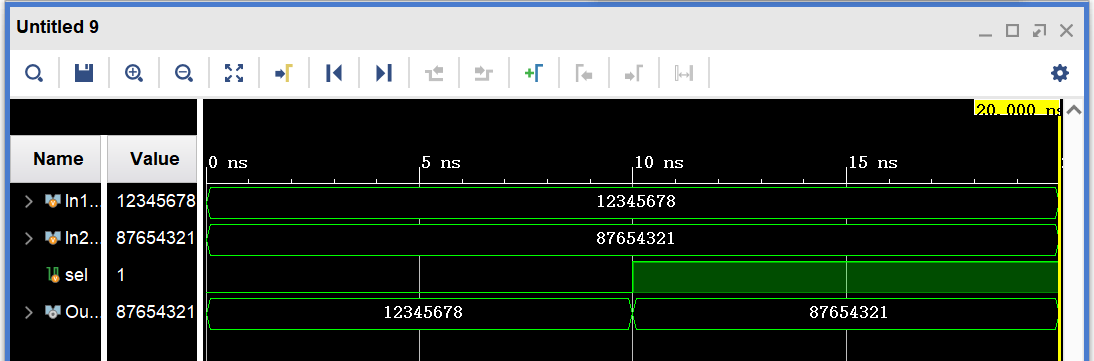
sel=1;

#10;

$stop;

end

endmodule



由波形图可知，sel为0时选择0通道的数据12345678输出，sel为1时选择1通道的数据87654321输出

**5、3 选 1 多路复用器 Mux3\_1**

`timescale 1ns / 1ps

module Mux3\_1sim();

reg [31:0] In1, In2,In3;

reg [1:0] sel;

wire [31:0] Out;

Mux3\_1 U0(In1, In2,In3, sel, Out);

defparam U0.n=32;

integer i;

initial

begin

In1=32'h12345678;

In2=32'h87654321;

In3=32'h33333333;

for (i=0;i<4;i=i+1)

begin

sel=i;

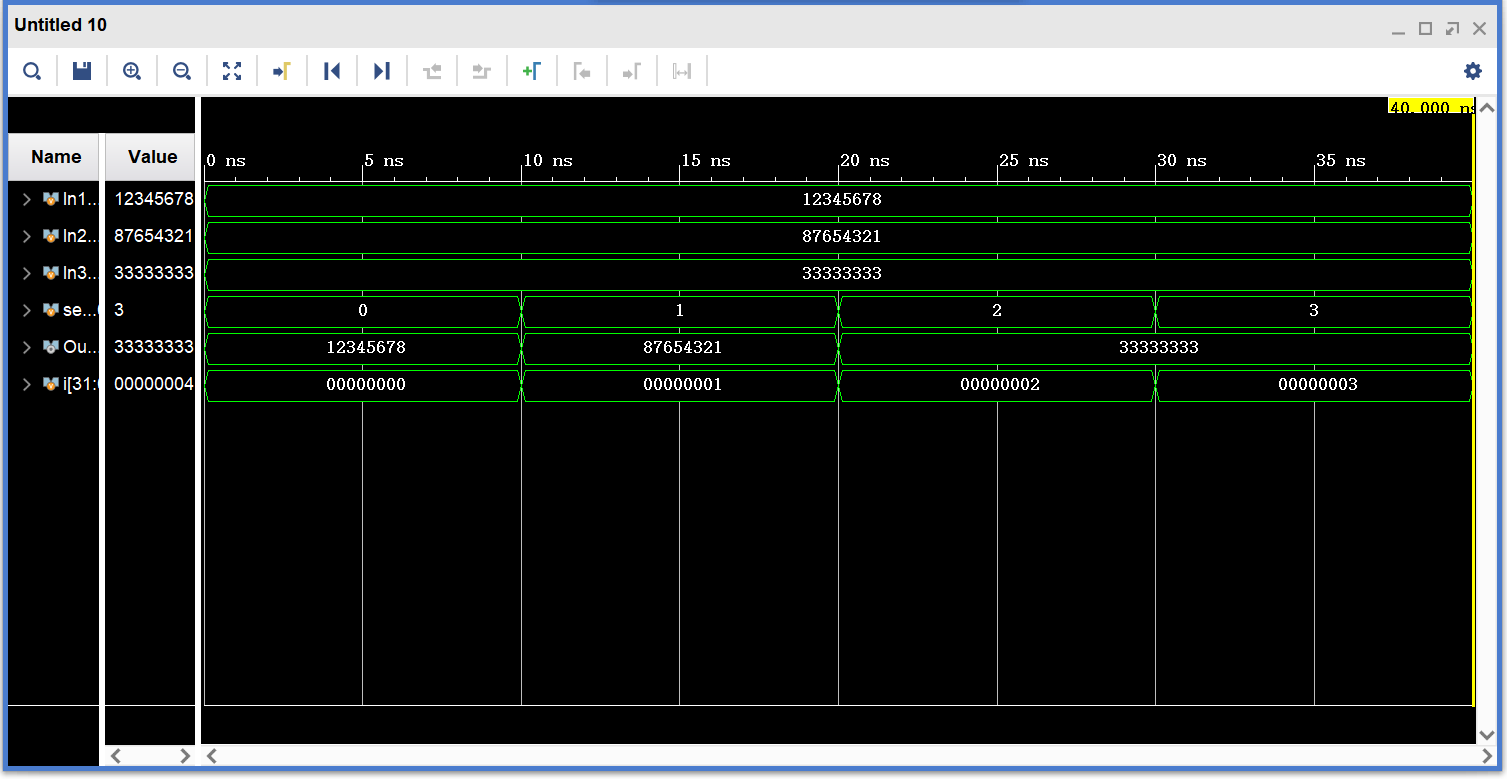
#10;

end

$stop;

end

endmodule



由波形可知，sel为0时选择0通道数据输出，sel为1时选择1通道数据输出sel为1x时选择通道2数据输出，符合其功能表

**6、左移运算器 LeftShift**

`timescale 1ns / 1ps

module Leftshiftsim( );

reg [31:0] In;

wire [31:0] Out0;

wire [27:0] Out1;

wire [28:0] Out2;

LeftShift U0(In, Out0);

defparam U0.n=32,U0.m=32,U0.x=2;

LeftShift U1(In[25:0], Out1);

defparam U1.n=26,U1.m=28,U1.x=2;

LeftShift U2(In[25:0], Out2);

defparam U2.n=26,U2.m=29,U2.x=3;

initial

begin

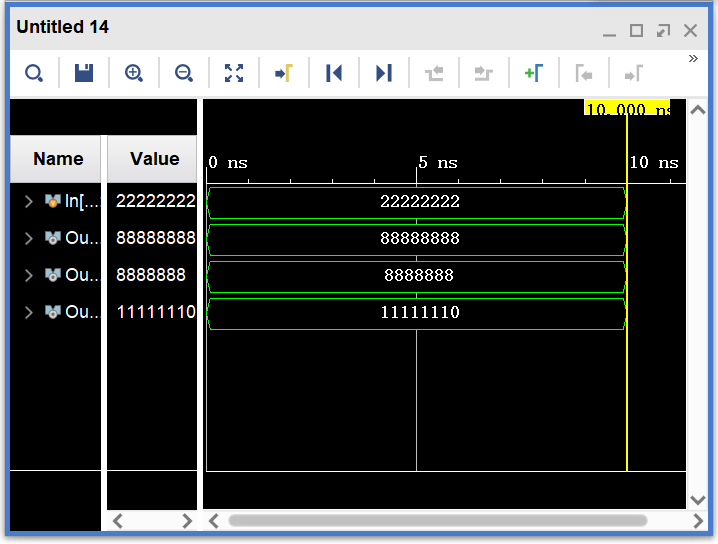
In=32'h22222222;

#10;

$stop;

end

endmodule



Out1是32位的左移2位，2乘以2的平方就是8，Out2是28位的左移2位，故转化为16进制后总共只有7位，Out3是左移三位，2的二进制是0010，左移3位后最低4位为0000，其他的为0001

**7、加法器 Adder**

`timescale 1ns / 1ps

module Addersim( );

reg [31:0] In1,In2;

wire [31:0] Out;

Adder U0(In1,In2,Out);

initial

begin

In1=4;

In2=8;

#10;

In1=32'h44444444;

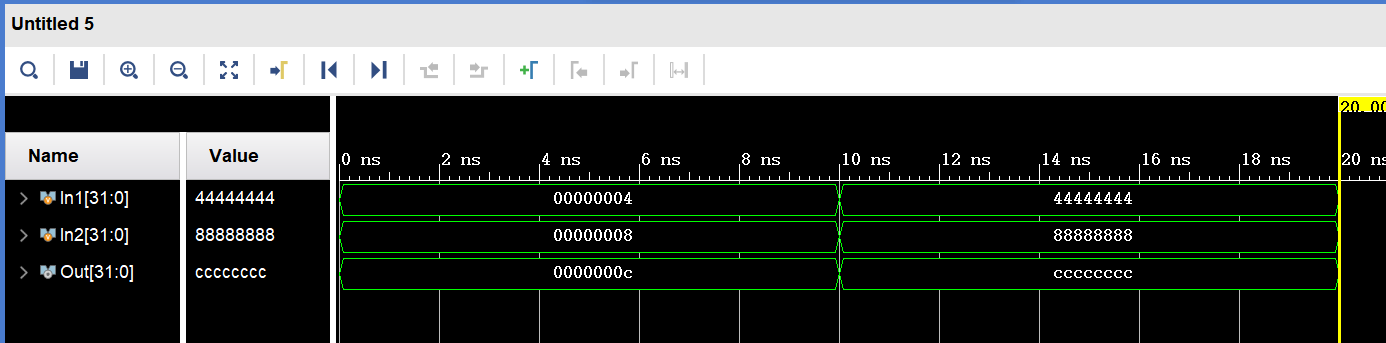
In2=32'h88888888;

#10;

$stop;

end

endmodule



4+8=12对应c，符合结果

**8、寄存器文件 RegFile**

`timescale 1ns / 1ps

module RegFileSim();

reg [4:0] RsAddr,RtAddr, WrAddr;

reg [31:0] DataIn;

reg RegWr,Clk;

wire [31:0] RsData, RtData;

RegFile U0\_RegFile(RsAddr, RtAddr, WrAddr, DataIn, RegWr,Clk, RsData, RtData);

parameter PERIOD = 10;

always begin

Clk = 1'b0;

#(PERIOD/2) Clk = 1'b1;

#(PERIOD/2);

end

integer i;

initial

begin

for(i=0;i<32;i=i+1)

U0\_RegFile.regs[i]=0;

RegWr=1;

for(i=0;i<10;i=i+1)

begin

WrAddr=i;

DataIn=i\*4;

#10;

end

RegWr=0;

for(i=0;i<10;i=i+1)

begin

RsAddr=i;

RtAddr=9-i;

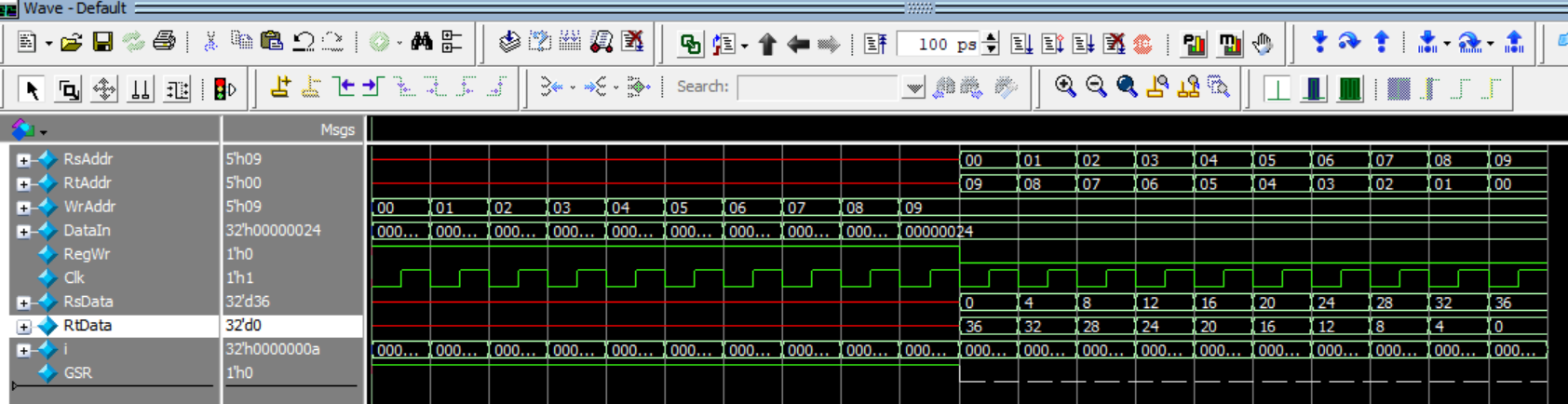
#10;

end

$stop;

end

endmodule



**9.符号扩展模块 SignedExtend**

`timescale 1ns / 1ps

module SignExtendsim( );

reg [15:0] In;

wire [31:0] Out;

SignedExtend U0(In, Out);

defparam U0.n=16,U0.m=32;

initial

begin

In=16'h1234;

#10;

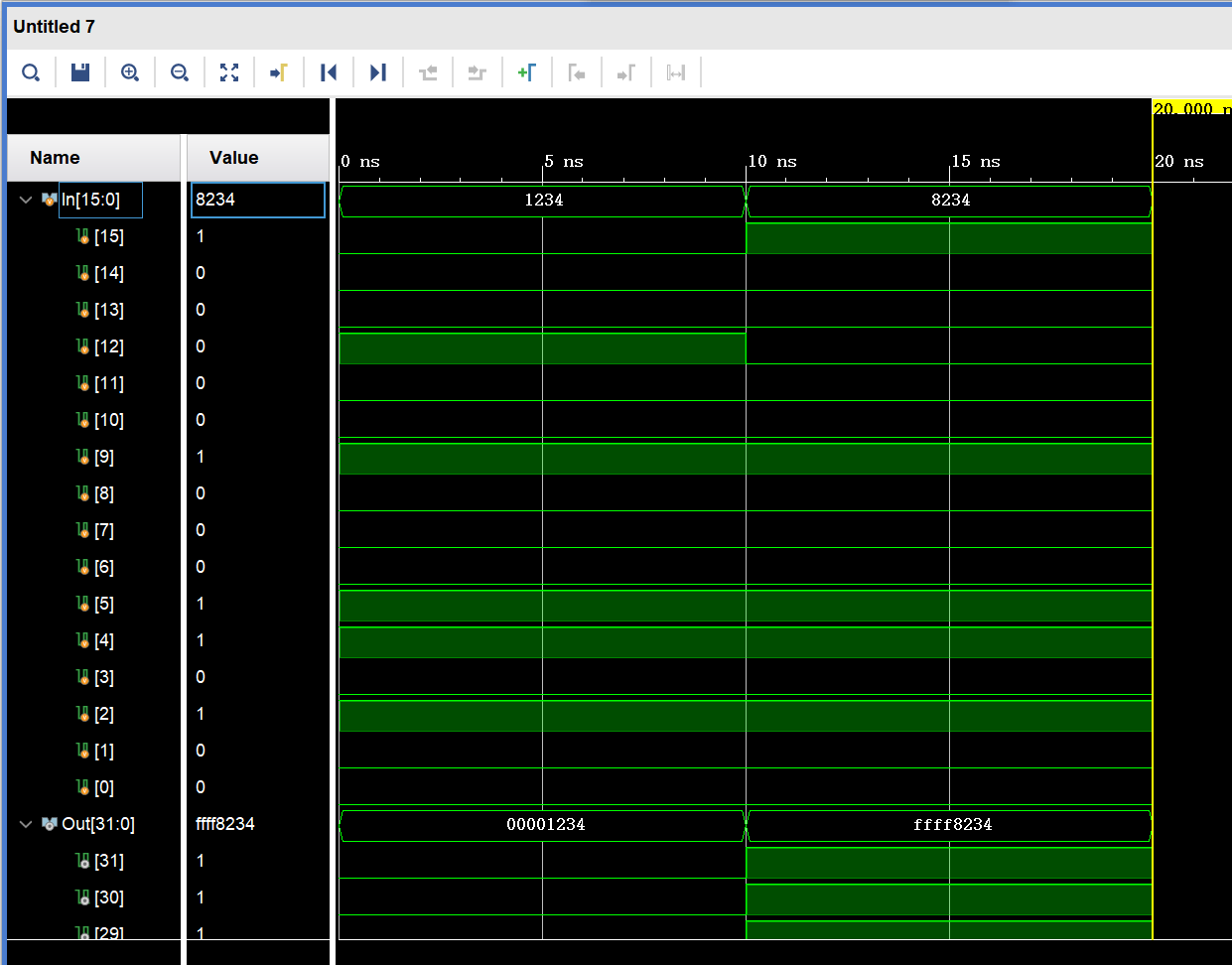
In=16'h8234;

#10;

$stop;

end

endmodule



1234的最高位为0，故扩展时最高位补0；8234的最高位为1，故扩展时最高位补1，扩展结果为ffff8234

**10、32 位宽算术逻辑运算单元 ALU**

`timescale 1ns / 1ps

module ALUSim();

reg signed [31:0] In1;

reg signed [31:0] In2;

reg [3:0] ALUCtr;

wire [31:0] ALURes;

wire Zero ;

ALU U0\_ALU(In1,In2,ALUCtr, ALURes,Zero);

initial

begin

In1 = 4;

In2 = 8;

ALUCtr = 0;

#10;

ALUCtr = 1;

#10;

ALUCtr = 2;

#10;

ALUCtr = 6;

#10;

ALUCtr = 7;

#10;

In1 = 32'h44444444;

In2 = 32'h88888888;

ALUCtr = 0;

#10;

ALUCtr = 1;

#10;

ALUCtr = 2;

#10;

ALUCtr = 6;

#10;

ALUCtr = 7;

#10;

In1 = 32'h99999999;

In2 = 32'h88888888;

ALUCtr = 0;

#10;

ALUCtr = 1;

#10;

ALUCtr = 2;

#10;

ALUCtr = 6;

#10;

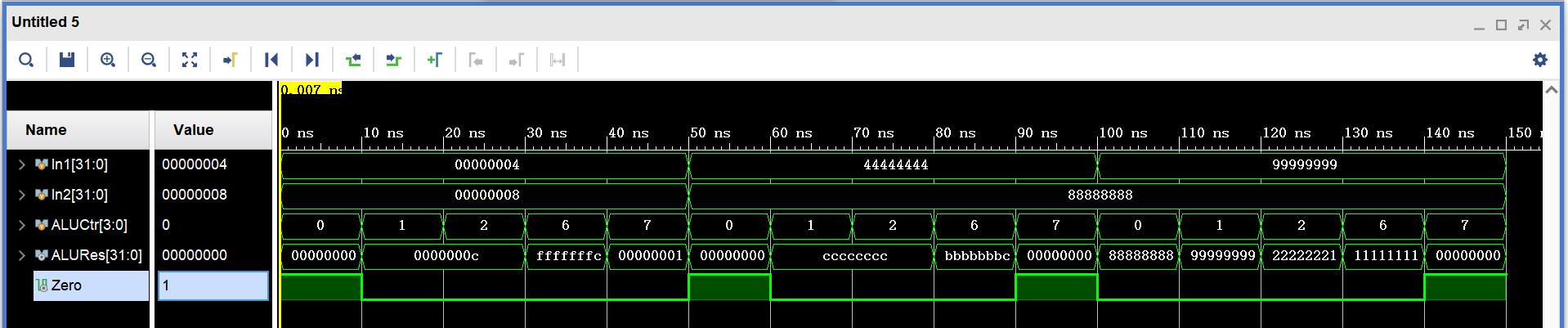
ALUCtr = 7;

#10;

$stop;

end

endmodule



ALUCtr为0,1,2,6,7时分别对应与，或，加，减，小于设置运算，4和8的与结果为0，或和加结果为12减的结果为-4，对应fffffffc，4<8，故小余运算结果为1，其他操作数运算情况也符合预期。

**11、位拼接器 Concat**

`timescale 1ns / 1ps

module Concatsim( );

reg [31:0] In1;

reg [31:0] In2;

wire [31:0] Out0;

wire [1:0] Out1;

Concat U0(In1[3:0], In2[27:0], Out0);

defparam U0.n=4,U0.m=28;

Concat U1(In1[0], In2[0], Out1);

defparam U1.n=1,U1.m=1;

initial

begin

In1=32'h12345679;

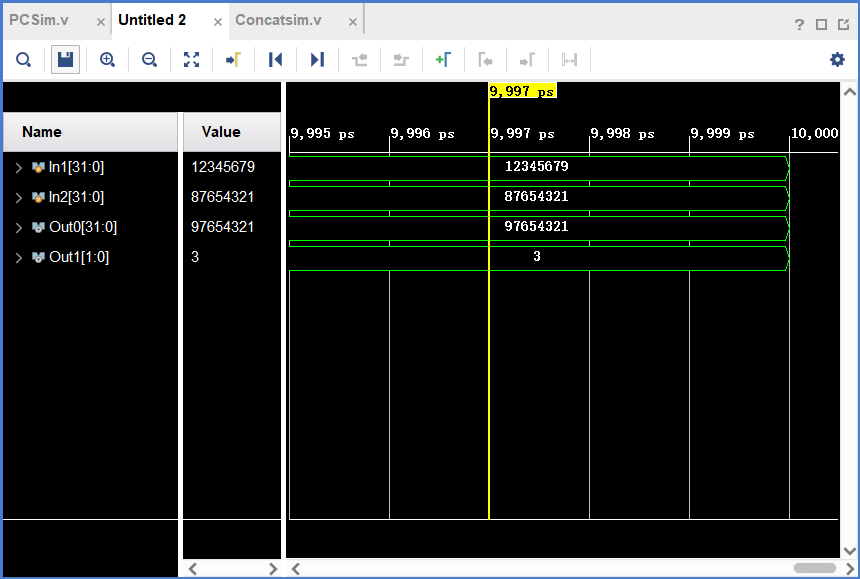
In2=32'h87654321;

#10;

$stop;

end

endmodule



第一个是把输入1低四位和输入2的高四位拼接，输出2是把输入1的最低位和输入2的最低位（都是1）拼接，得到3，即11，符合预期

**12、数据存储器 DataRAM**

`timescale 1ns / 1ps

module DataRamsim( );

reg [4:0] Addr;

reg [31:0] DataIn;

wire [31:0] DataOut;

reg MemWR,Clk;

DataRAM U0\_RAM(Addr,DataIn,MemWR,Clk,DataOut);

defparam U0\_RAM. n=5,U0\_RAM. m=32;

parameter PERIOD = 10;

always begin

Clk = 1'b0;

#(PERIOD/2) Clk = 1'b1;

#(PERIOD/2);

end

integer i;

initial

begin

MemWR = 1;

for(i=0;i<10;i=i+1)

begin

Addr=i;

DataIn=i\*4;

#10;

end

MemWR=0;

for(i=0;i<10;i=i+1)

begin

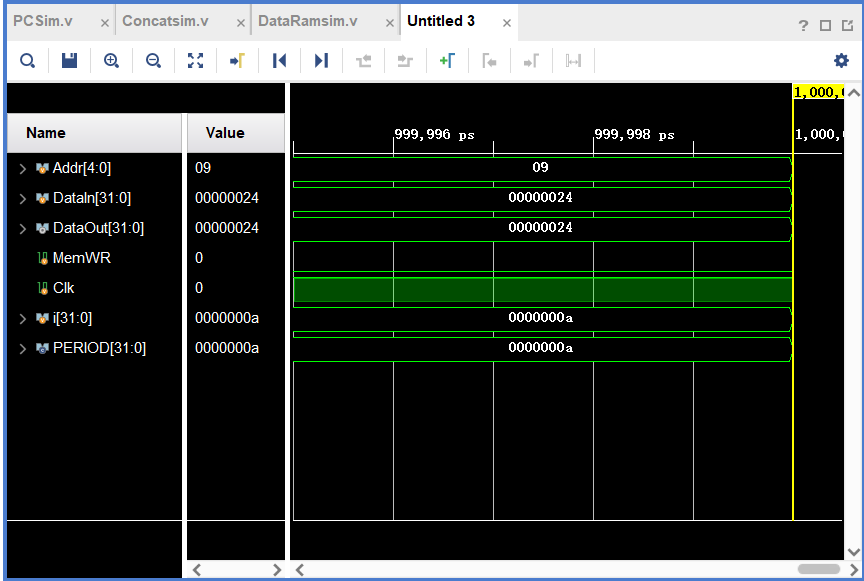
Addr=i;

#10;

end

end

endmodule



**13、整体模块实现MIPSCPU 简单指令集MIPS微处理器**

`timescale 1ns / 1ps

module mipscpusim( );

reg Clk, Reset;

MIPSCPU uut(Clk, Reset);

**parameter period=10;**

**always begin //产生时钟信号**

**Clk=1'b0;**

**#(period/2) Clk = 1'b1;**

**#(period/2);**

**end**

integer i;

initial

begin

//Reset = 1;

//#10;

**$readmemh ("D:/SOFTWARES/FPGA\_Projects/MIPS/MIPS\_CPU\_S/machinecode.txt", uut.ROM\_U0.regs, 0, 11); //读取机器指令**

for(i=0;i<32;i=i+1)

begin **//RegFile 以及 DataRAM 前 32 个存储单元存储的初始值为单元地址\*4**

**uut.RegFile\_U1.regs[i]=i\*4;**

**uut.RAM\_U3.regs[i]=i\*4;**

end

Reset = 0;

#10;

Reset = 1;

#10;

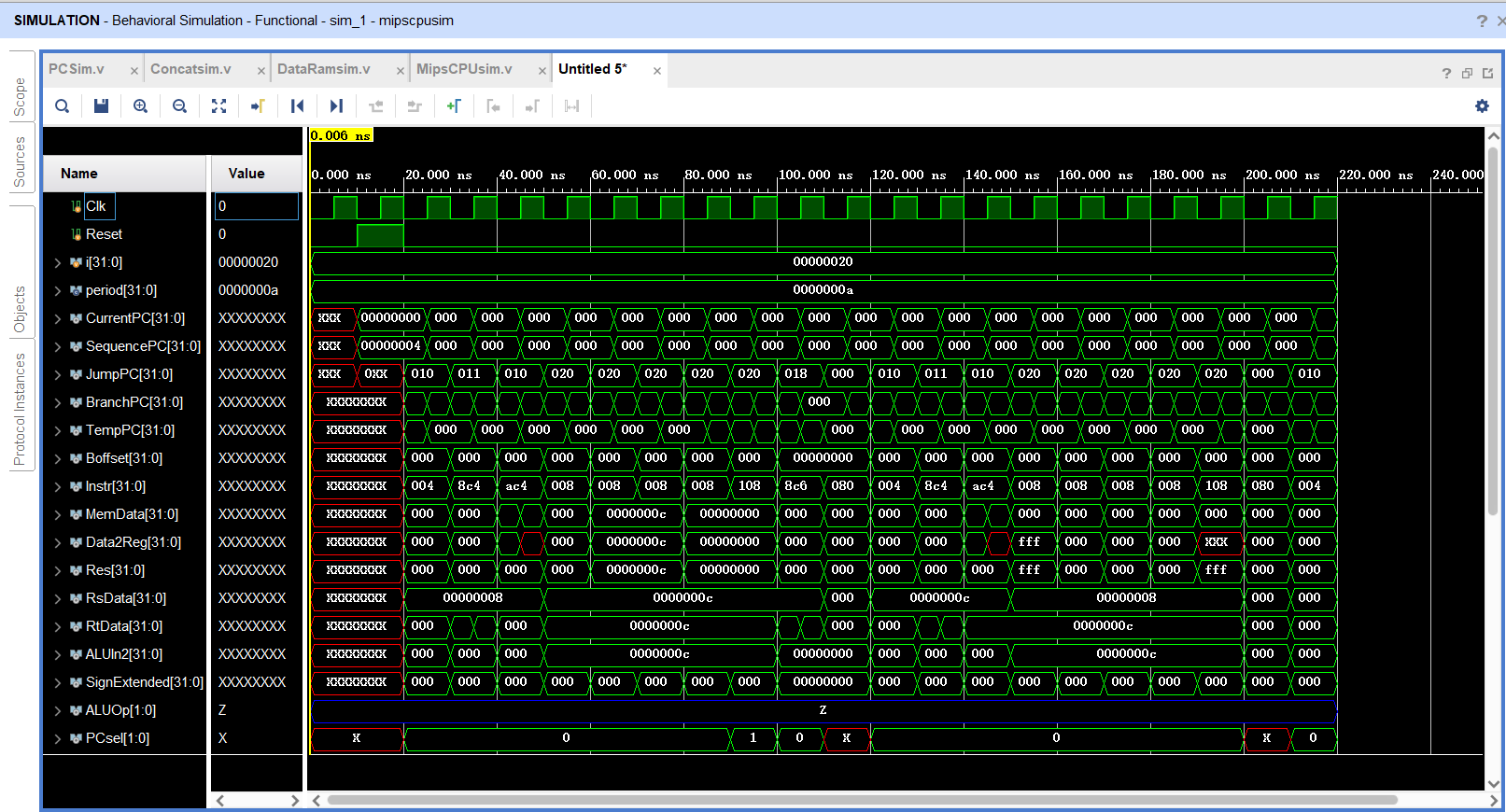
Reset = 0;

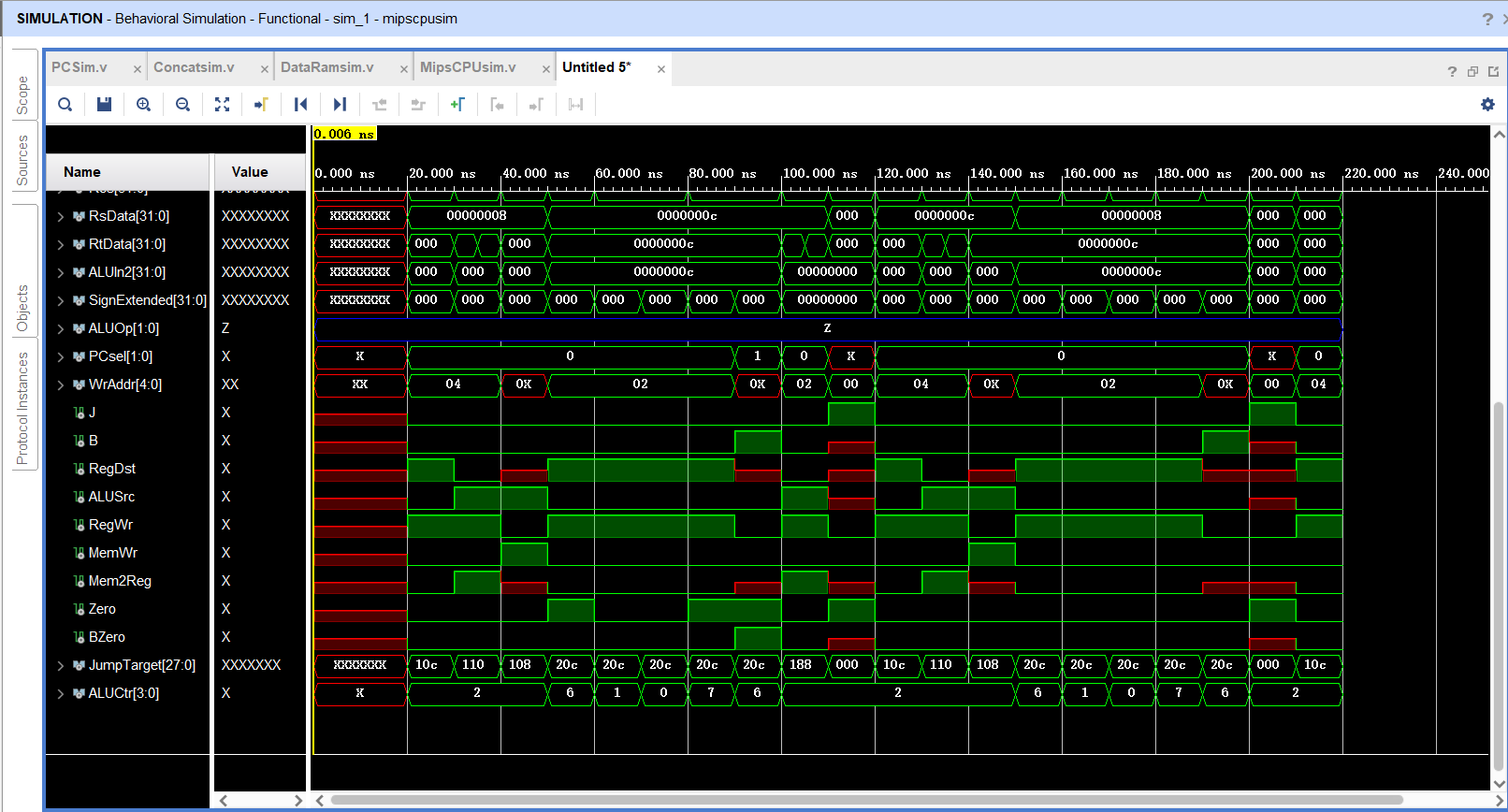
#200;

$stop;

end

endmodule





InstrROM 从地址 0 开始的连续 11 个存储单元存放机器指令如文件 test\_readmemh.coe 文件，test\_readmemh.coe 文件内容如下：

00432020

8c440004

ac420008

00831022

00831025

00831024

0083102a

10830001

08000000

8c620000

08000000

对应汇编指令如下：

main:

add $4,$2,$3 #0

lw $4,4($2) #1

sw $2,8($2) #2

sub $2,$4,$3 #3

or $2,$4,$3 #4

and $2,$4,$3 #5

slt $2,$4,$3 #6

beq $4,$3,exit #7

j main #8

exit: lw $2,0($3) #9

j main #10

前面7条指令都是顺序往下直行，到第七条指令时符合条件则跳转。$3为复位时的初始值0,$4为从数据存储器装载的值，若数据存储器地址为4的存储字的初始值不为0，则顺序执行所有指令之后在0~8号指令之间循环；若数据存储器地址为4的存储字初始值为0，则跳过第8条指令，进行到9号指令，再循环。

观察PC的变化可知，PC先从0,4,8,12,16,20,24,28，前7条指令顺序执行，数据存储器第0号地址的存储字初始值为0，执行第7条指令的时候Zero的值为1，说明两寄存器的值相等，符合跳转条件，故跳过了第8条指令，进入了第9条指令，也就是地址变为36。执行完9,10条指令后又返回原循环。

之后指令执行的过程中，由于RegFile 以及 DataRAM 前 32 个存储单元存储的初始值为单元地址\*4，都不为0，所以程序在0~7条语句之间循环，PC在0,4,8，12,16,20，24,28之间循环跳转。

第一条指令为R型指令，高6位为000000，低6位功能码为100000，译码译出的ALUCtr 为0010，即2，执行加运算指令，波形符合预期。

第二条指令为I型指令，高6位为100011,译码译出的ALUCtr 也为0010，对应lw指令。低16位Imm域为常数地址0x0004，符号扩展后也为4，在ALU中进行加运算，得到的地址在数据存储器中读到相应数据再写到寄存器文件中。

第三条指令为I型指令，高6位为101011,译码译出的ALUCtr 也为0010，对应sw指令。低16位Imm域为常数地址0x0008，符号扩展后也为8，在ALU中进行加运算，得到地址，再在数据存储器把数据写到相应的地址中。

第4,5,6条指令也为R型指令，低6位功能码不同，分别执行sub,or,and,slt不同功能。

第七条指令为I型指令，因为zero为1，所以跳转到exit语句的位置。

第八条指令为J型指令，无条件跳转到main语句也就是第0条语句的位置，所以PC又恢复为0。

第一条指令的Rs对应$2，Rt对应$3,RsData=8=2\*4,符合地址都是2的整数倍，RtData=0xc=12=3\*4,相加后的结果为0x14=20=8+12,符合；

1. 三条指令的Rs都是$2，二的偏移地址为4，三的偏移地址为8，对应的SignExtended分别为4和8，波形符合预期。

之后的sub,or,and,slt都为R型指令，由于$4与$3对应的值相等，都为0xc，他们相减后为0，相与、相或之后的结果相同，左移之后右边补0，所以最终得到的是0.

再执行beq指令，$4,$3寄存器对应的值相等，Zero输出的值为1，跳转到

exit指令处，再往下执行一条 j main ，之后回到第0条指令。

**六、实验心得**

在这次实验中，我用verilog HDL语言设计了一个MIPS单周期微处理器。我重新复习了一遍MIPS微处理器的相关知识，更加深入地理解了MIPS单周期微处理器的运行原理。加深了对MIPS微处理器的基本结构与其每个模块工作过程的理解，也理解了软件控制硬件的工作原理。经过长时间与VIVADO的搏斗，我对它的使用也更加熟悉，尤其是理解了Vivado中各种模块是如何联系在一起构成一整个系统的。这次实验收获非常大！