**电子科技大学**

**计算机科学与工程学院**

**标 准 实 验 报 告**

**（实验）课程名称： 单周期CPU的设计与实现**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：王正仁 学 号：2019081308021 指导教师：陈虹**

**实验地点：主楼 A2-411 实验时间：第 14,16-18 周周六（5-8 节）**

1. **实验室名称：**

A2-411

1. **实验项目名称：**

单周期CPU的设计与实现

1. **实验学时：**

实验学时共16学时。

1. **实验原理：**

本实验中的单周期CPU的工作原理如图4-1所示。

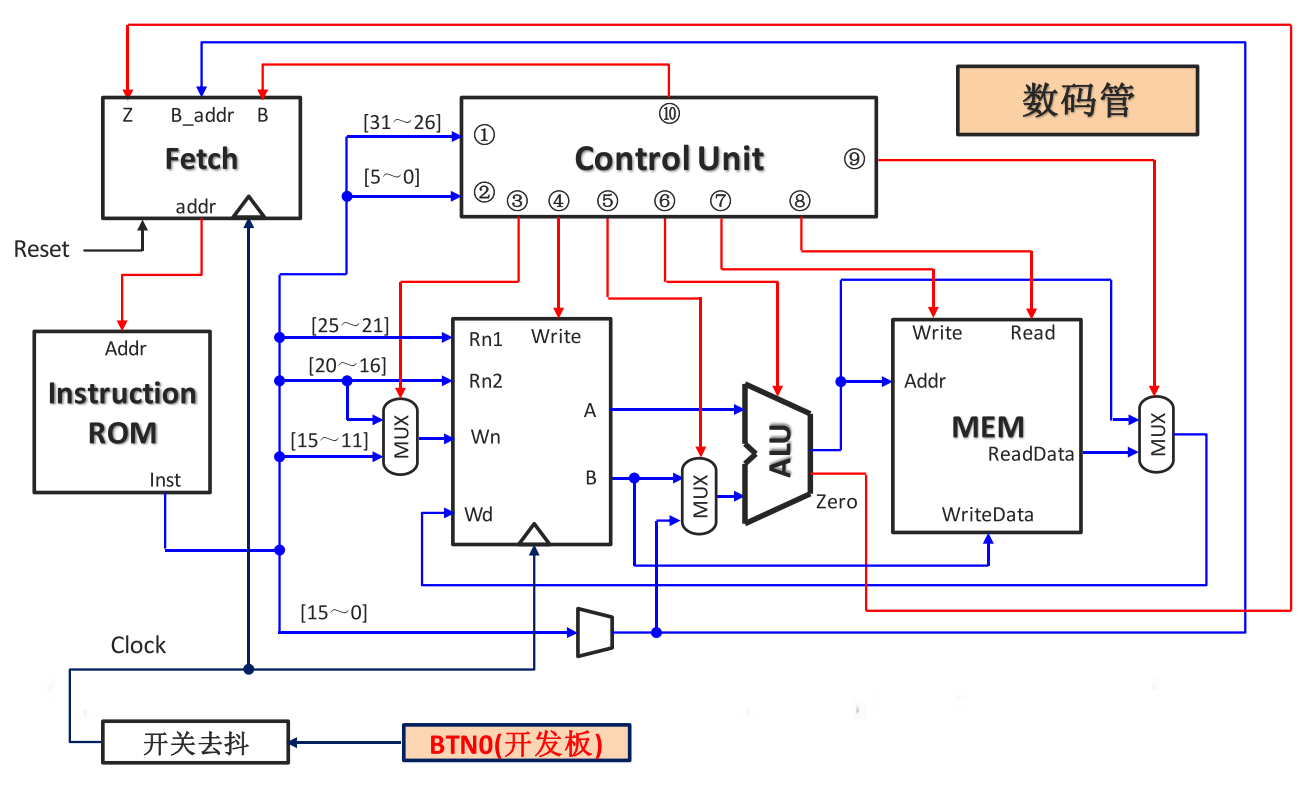


图4-1 单周期CPU总体电路

一条指令的执行过程包括：取指令→分析指令→执行指令→保存结果（如果有的话）。对于单周期CPU来说，这些执行步骤均在一个时钟周期内完成。

1. 本实验的所设计的指令共有10条，分为3种类型：R型、I型和J型。不同的指令类型有不同的数据路径。
2. 在每个时钟周期根据当前PC的内容取出一条指令，将指令的op和func送控制器（Control Unit）译码，产生指令执行的控制码和运算码，控制指令的执行。
3. 寄存器堆、多路选择器、数据存储器及运算器等部件在相应控制码的控制下工作，产生指令的输出数据流，并将结果写入相应的存储部件。
4. 在下一个时钟周期到来时，重复2、3步骤，直到程序执行完成。
5. **实验目的**
6. 通过本实验，让学生掌握单周期CPU的工作原理，控制器、运算器等部件设计的基本方法和技能，加深对所学知识的理解和掌握。
7. 通过使用硬件描述语言Verilog、EDA工具软件进行软件设计与仿真，并在FPGA上实现，以培养学生的分析和设计CPU的能力。
8. **实验内容**
9. 拟定本实验的指令系统，指令应有计算类型、访问存储器类型、条件转移类型和无条件转移等，指令数应不少于10条。
10. CPU各功能模块的设计与实现。
11. CPU的封装。
12. 对所做设计的仿真测试和板级验证。
13. **实验器材（设备、元器件）：**
14. 安装了Xilinx ISE Design Suite 13.4的PC机一台
15. FPGA开发板：Anvyl Spartan6/XC6SLX45
16. 计算机与FPGA开发板通过JTAG（Joint Test Action Group）接口连接，其连接方式如图7-1所示。

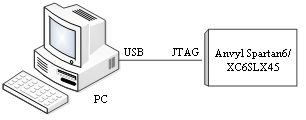


图7-1 PC计算机与FPGA开发板的连接

1. **实验步骤**
2. 拟定本实验所设计CPU的指令系统

本实验所设计的CPU一共支持10条MIPS指令，包含了R型、I型和J型指令。如表8-1所示，其中R型指令为Add, Sub, And, Or, Xor；I型指令为Lw, SW, Beq, Lui；J型指令为J。

表8-1 本实验所涉及的10条MIPS指令

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R型指令 | | | | | | | |
| 指令 | [31:26] | [25:21] | [20:16] | [15:11] | [10: 6] | [5:0] | 功能 |
| Add | 000000 | rs | rt | rd | 000000 | 100000 | 寄存器加 |
| Sub | 000000 | rs | rt | rd | 000000 | 100010 | 寄存器减 |
| And | 000000 | rs | rt | rd | 000000 | 100100 | 寄存器与 |
| Or | 000000 | rs | rt | rd | 000000 | 100101 | 寄存器或 |
| Xor | 000000 | rs | rt | rd | 000000 | 100110 | 寄存器异或 |
| I型指令 | | | | | | | |
| Lw | 100011 | rs | rt | offset | | | 取数据 |
| Sw | 101011 | rs | rt | offset | | | 存数据 |
| Beq | 000100 | rs | rt | offset | | | 相等转移 |
| Lui | 001111 | 00000 | rt | immediate | | | 设置高位 |
| J型指令 | | | | | | | |
| J | 000010 | address | | | | | 跳转 |

1. 新建工程（New Project）

启动ISE Design Suite 13.4软件，然后选择菜单File→New Project，弹出New Project Wizard对话框，在对话框中输入工程名Single\_Cycle\_CPU，并指定工作路径D:\Single\_Cycle\_CPU。

1. 基本功能器件的设计与实现
2. 32位2选1多路选择器的设计与实现

32位2选1选择器有三个输入端，分别为两个32位的数据输入端A和B、一个1位的输入选择信号Sel，同时有一个32位的数据输出端O。由Sel信号决定输出A或B：Sel=1时，输出为B；Sel=0时，输出为A。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：MUX32\_2\_1，然后输入其实现代码：

module MUX32\_2\_1(

input [31:0] A,

input [31:0] B,

input Sel,

output[31:0] O

);

assign O = Sel? B : A;

endmodule

代码8-1 模块MUX32\_2\_1的实现

在ISE集成开发环境中，对模块MUX32\_2\_1进行综合（Synthesize），综合结果如图8-1所示。

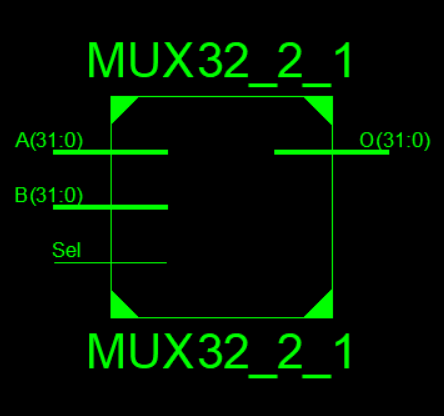


图8-1 模块MUX32\_2\_1的综合结果

在ISE集成开发环境中，对模块MUX32\_2\_1进行仿真（Simulation）。首先输入如下测试代码：

initial begin

// Initialize Inputs

A = 0;

B = 0;

Sel = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

A <= 32'hFFFF0000;

B <= 32'h0000FFFF;

Sel <= 1'b0;

#100;

Sel <= 1'b1;

end

代码8-2 模块MUX32\_2\_1的激励

然后进行仿真，仿真结果如图8-2所示。在0~100ns时，A=0x00000000，B=0x00000000，O=0x00000000，Sel=0；在100~200ns时，A=0xffff0000，B=0x0000ffff，O=0xffff0000，Sel=0；在200~300ns时，A=0xffff0000，B=0x0000ffff，O=0x0000ffff，Sel=1。



图8-2 模块MUX32\_2\_1的仿真结果

1. 5位2选1多路选择器的设计与实现

由于通用寄存器地址长度为5位，所以也需要实现5位2选1多路选择器。同样地，5位2选1选择器有三个输入端，分别为两个5位的数据输入端A和B、一个1位的输入选择信号Sel，同时有一个5位的数据输出端O。由Sel信号决定输出A或B：Sel=1时，输出为B；Sel=0时，输出为A。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：MUX5\_2\_1，然后输入其实现代码：

module MUX5\_2\_1(

input [4:0] A,

input [4:0] B,

input Sel,

output[4:0] O

);

assign O = Sel? B : A;

endmodule

代码8-3 模块MUX5\_2\_1的实现

在ISE集成开发环境中，对模块MUX5\_2\_1进行综合（Synthesize），综合结果如图8-3所示。

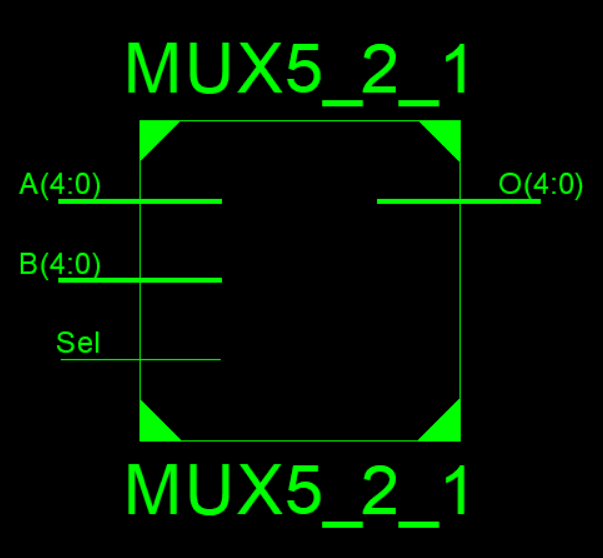


图8-3 模块MUX5\_2\_1的综合结果

在ISE集成开发环境中，对模块MUX5\_2\_1进行仿真。首先输入如下测试代码：

initial begin

// Initialize Inputs

A = 0;

B = 0;

Sel = 0;

// Wait 100 ns for global reset to finish

#100;

A = 5'b11111

B = 5'b00000

Sel = 0;

#100

Sel = 1;

end

代码8-4 模块MUX5\_2\_1的激励

然后进行仿真，仿真结果如图8-4所示。在0~100ns时，A=0b00000，B=0b00000，O=0b00000，Sel=0；在100~200ns时，A=0b11111，B=0b00000，O=0b11111，Sel=0；在200~300ns时，A=0b11111，B=0b00000，O=0b00000，Sel=1。

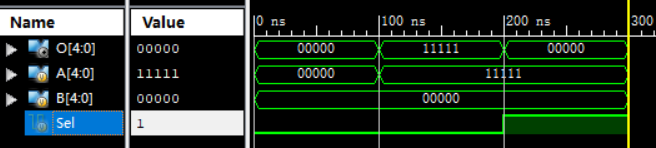


图8-4 模块MUX5\_2\_1的仿真结果

1. 30位2选1多路选择器的设计与实现

由于拓展题要求取指部件内部使用30位PC，所以也需要实现30位2选1多路选择器。同样地，30位2选1选择器有三个输入端，分别为两个30位的数据输入端A和B、一个1位的输入选择信号Sel，同时有一个30位的数据输出端O。由Sel信号决定输出A或B：Sel=1时，输出为B；Sel=0时，输出为A。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：MUX30\_2\_1，然后输入其实现代码：

module MUX30\_2\_1(

input [29:0] A,

input [29:0] B,

input Sel,

output[29:0] O

);

assign O = Sel? B : A;

endmodule

代码8-5 模块MUX30\_2\_1的实现

在ISE集成开发环境中，对模块MUX30\_2\_1进行综合，综合结果如图8-5所示。

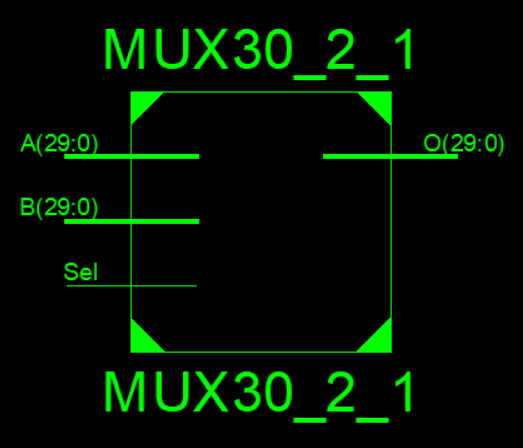


图8-5 模块MUX30\_2\_1的综合结果

在ISE集成开发环境中，对模块MUX30\_2\_1进行仿真。首先输入如下测试代码：

initial begin

// Initialize Inputs

A = 0;

B = 0;

Sel = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

A <= 30'h3FFF0000;

B <= 30'h0000FFFF;

Sel <= 1'b0;

#100;

Sel <= 1'b1;

end

代码8-6 模块MUX30\_2\_1的激励

然后进行仿真，仿真结果如图8-6所示。在0~100ns时，A=0x00000000，B=0x00000000，O=0x00000000，Sel=0；在100~200ns时，A=0x3fff0000，B=0x0000ffff，O=0x3fff0000，Sel=0；在200~300ns时，A=0x3fff0000，B=0x0000ffff，O=0x0000ffff，Sel=1。

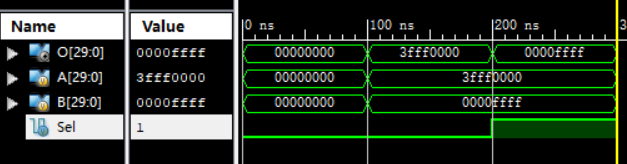


图8-6 模块MUX30\_2\_1的仿真结果

1. 左移2位器的设计与实现

由于MIPS32架构中指令字长为4字节，故地址低两位始终为0，在32位PC的系统中需要使用左移2位器。左移2位器有一个32位输入端d和一个32位输出端o。功能为将d左移2位，即将o赋值为d的低30位后拼接上2位0。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Left\_2\_Shifter，然后输入其实现代码：

module Left\_2\_Shifter(

input [31:0] d ,

output [31:0] o

);

assign o = { d[29:0] , 2'b00};

endmodule

代码8-7 模块Left\_2\_Shifter的实现

在ISE集成开发环境中，对模块Left\_2\_Shifter进行综合，综合结果如图8-7所示。

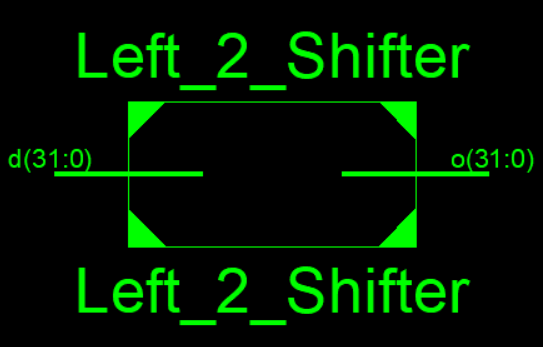


图8-7 模块Left\_2\_Shifter的综合结果

在ISE集成开发环境中，对模块Left\_2\_Shifter进行仿真。首先输入如下测试代码：

initial begin

d = 0;

#100;

d=32'h00000001;

#100;

d=32'hc0000000;

end

代码8-8 模块Left\_2\_Shifter的激励

然后进行仿真，仿真结果如图8-8所示。在0~100ns时，d=0x00000000，o=0x00000000；在100~200ns时，d=0x00000001，o=0x00000004；在200~300ns时，d=0xc0000000，o=0x00000000。

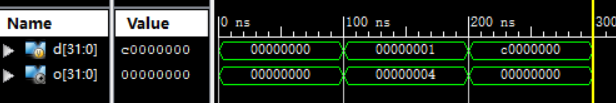


图8-8 模块Left\_2\_Shifter的仿真结果

注：由于本人完成了拓展要求——使用30位的PC寄存器，因此系统中无需左移器、32位加法器等相关器件。下文将略去与32位PC相关的器件。

1. 30位加法器的设计与实现

在30位取指令部件中，由于相对寻址，自增等的功能要求，需要使用30位加法器。30位加法器有两个30位输入端A和B，一个32位输出端C。功能即输出C=A+B，为组合逻辑器件。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：ADD30，然后输入其实现代码：

module ADD30(

input [29:0] A , B ,

output [29:0] C

);

assign C = A + B;

endmodule

代码8-9 模块ADD30的实现

在ISE集成开发环境中，对模块ADD30进行综合，综合结果如图8-9所示。

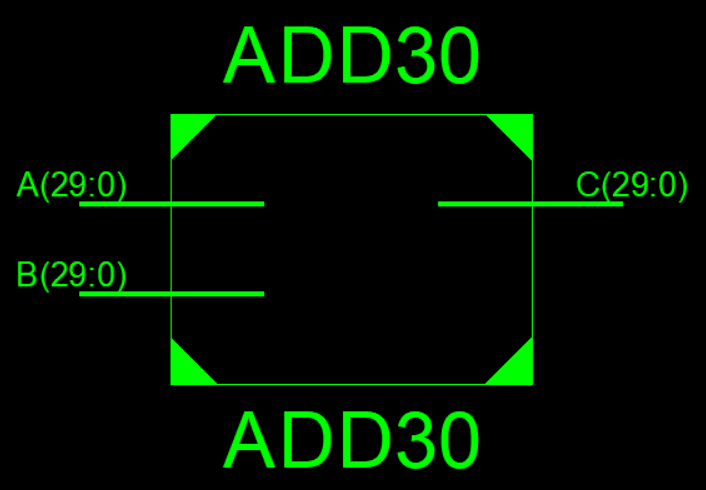


图8-9 模块ADD30的综合结果

在ISE集成开发环境中，对模块ADD30进行仿真。首先输入如下测试代码：

initial begin

A = 30'h00000001;

B = 30'h00000001;

#100;

A = 30'h3FFF0000;

B = 30'h0000FFFF;

#100;

A = 30'h3FFF0001;

B = 30'h0000FFFF;

end

代码8-10 模块ADD30的激励

然后进行仿真，仿真结果如图8-10所示。在0~100ns时，A=0x00000001，B=0x00000001，C=0x00000010；在100~200ns时，A=0x3fff0000，B=0x0000ffff，O=0x3fffffff；在200~300ns时，A=0x3fff0001，B=0x0000ffff，O=0x00000000。



图8-10 模块ADD30的仿真结果

1. 16-32位符号扩展器的设计与实现

由于偏移寻址，I型运算指令等的功能要求，需要使用16-32位符号扩展器。16-32位符号扩展器有一个16位的数据输入端a和一个32的数据输出端b，用于将输入的16位的数据符号扩展为32位的数据。规则为：若a的符号位为0，则在最高位前补16个0；若a的符号位为1，则在最高位前补16个1。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Sign\_Extender，然后输入其实现代码：

module Sign\_Extender(

input [15:0] a ,

output [31:0] b

);

assign b = {a[15] ? 16'hffff : 16'h0 , a};

endmodule

代码8-11 模块Sign\_Extender的实现

在ISE集成开发环境中，对模块Sign\_Extender进行综合，综合结果如图8-11所示。

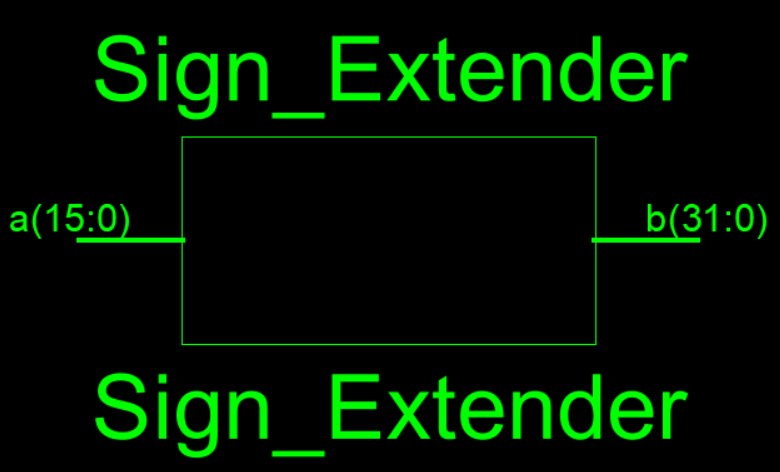


图8-11 模块Sign\_Extender的综合结果

在ISE集成开发环境中，对模块Sign\_Extender进行仿真。首先输入如下测试代码：

initial begin

a = 0;

#100;

a = 16'h7fff;

#100;

a = 16'h8fff;

end

代码8-12 模块Sign\_Extender的激励

然后进行仿真，仿真结果如图8-12所示。在0~100ns时，a=0x0000，b=0x00000000；在100~200ns时，a=0x7fff，b=0x00007fff；在200~300ns时，a=0x8fff，b=0xffff8fff。

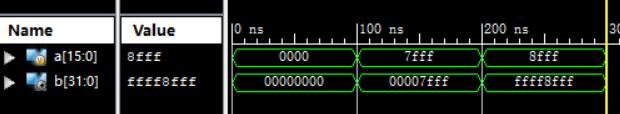


图8-12 模块Sign\_Extender的仿真结果

1. 16-30位符号扩展器的设计与实现

由于PC相对寻址的功能要求，需要使用16-30位符号扩展器。同样地，16-30位符号扩展器有一个16位的数据输入端a和一个30的数据输出端b，用于将输入的16位的数据符号扩展为30位的数据。规则为：若a的符号位为0，则在最高位前补14个0；若a的符号位为1，则在最高位前补14个1。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Sign\_Extender\_30，然后输入其实现代码：

module Sign\_Extender\_30(

input [15:0] a ,

output [29:0] b

);

assign b = {a[15] ? 14'b11111111111111 : 14'b0 , a} ;

endmodule

代码8-13 模块Sign\_Extender\_30的实现

在ISE集成开发环境中，对模块Sign\_Extender\_30进行综合，综合结果如图8-13所示。

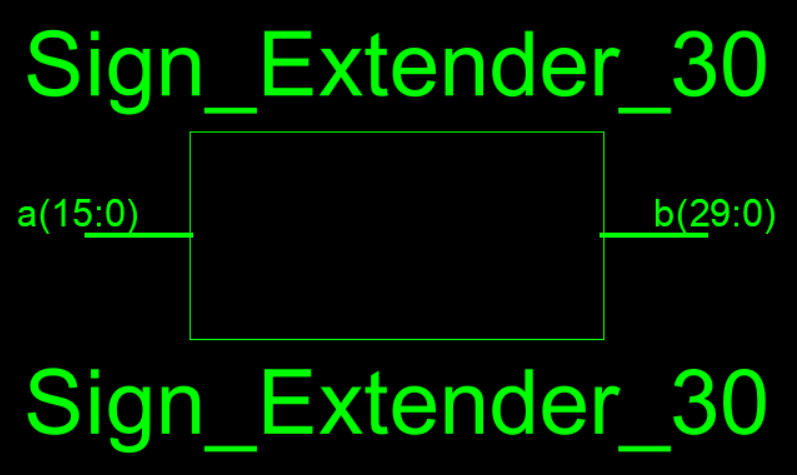


图8-13 模块Sign\_Extender\_30的综合结果

在ISE集成开发环境中，对模块Sign\_Extender\_30进行仿真。首先输入如下测试代码：

initial begin

a = 0;

#100;

a = 16'h7fff;

#100;

a = 16'h8fff;

end

代码8-14 模块Sign\_Extender\_30的激励

然后进行仿真，仿真结果如图8-14所示。在0~100ns时，a=0x0000，b=0x00000000；在100~200ns时，a=0x7fff，b=0x00007fff；在200~300ns时，a=0x8fff，b=0x3fff8fff。

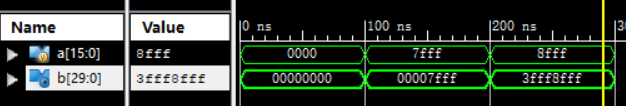


图8-14 模块Sign\_Extender\_30的仿真结果

1. 运算器（ALU）的设计与实现

ALU，即算术逻辑单元，用于执行算数运算和逻辑运算。输入包括两个32位的操作数输入端A和B，和一个3位的控制ALU执行具体功能的控制端ALU\_op。输出包括一个32位的运算结果Result和一个零标志位Zero，即当 Result=0时，Zero=1；否则Zero=0。如表8-2所示，根据不同的ALU\_op值，ALU会执行不同的操作。

表8-2 ALU\_op对应的ALU功能

|  |  |
| --- | --- |
| ALU\_op | ALU功能 |
| 000 | Result=A+B |
| 100 | Result=A-B |
| 001 | Result=A&B |
| 101 | Result=A|B |
| 010 | Result=A⊕B |
| 110 | Result高16位置为b，低16位清0 |

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：ALU，然后输入其实现代码：

module ALU(

input [31:0] A,B, //输入数据

input [2:0] ALU\_op, //控制信号

output [31:0] Result, //输出数据

output Zero //零标志位

);

assign Result = (ALU\_op == 3'b000) ? A + B :

(ALU\_op == 3'b100) ? A - B :

(ALU\_op == 3'b001) ? A & B :

(ALU\_op == 3'b101) ? A | B :

(ALU\_op == 3'b010) ? A ^ B :

(ALU\_op == 3'b110) ? {B[15:0],16'h0}:32'hxxxxxxxx;

assign Zero = ~| Result;

//~|为规约操作符,将各位相或后取反

endmodule

代码8-15 模块ALU的实现

在ISE集成开发环境中，对模块ALU进行综合，综合结果如图8-15所示。

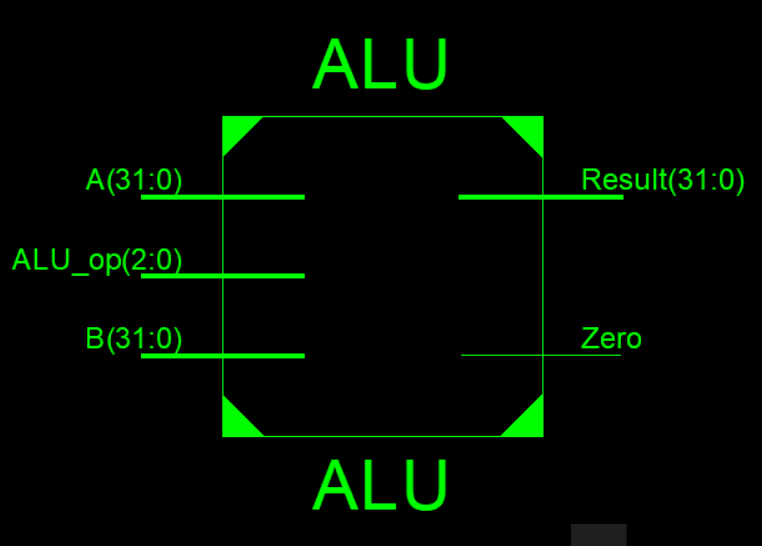


图8-15 模块ALU的综合结果

在ISE集成开发环境中，对模块ALU进行仿真。首先输入如下测试代码：

initial begin

A = 32'h45;

B = 32'h45;

ALU\_op = 3'b000; //add

#100

ALU\_op = 3'b100; //sub

#100

ALU\_op = 3'b001; //and

#100

ALU\_op = 3'b101; //or

#100

ALU\_op = 3'b010; //xor

#100

ALU\_op = 3'b110; //lui

end

代码8-16 模块ALU的激励

然后进行仿真，仿真结果如图8-16所示。维持A=B=0x00000045，在0~100ns时ALU\_OP=0（加法），Result=0x0000008a，Zero=0；在100~200ns时，ALU\_OP=4（减法），Result=0x00000000，Zero=1；在200~300ns时，ALU\_OP=1（位与），Result=0x00000045，Zero=0；在300~400ns时，ALU\_OP=5（位或），Result=0x00000045，Zero=0；在400~500ns时，ALU\_OP=2（异或），Result=0x00000000，Zero=1；在500~600ns时，ALU\_OP=6（置高位），Result=0x00450000，Zero=0。

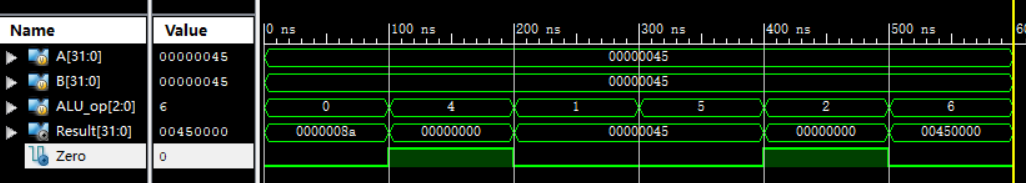


图8-16 模块ALU的仿真结果

1. 寄存器堆（Register File）的设计与实现

寄存器堆，即寄存器组，由32个32位寄存器构成，用于数据的快速读写。其输入包括两个5位的被读寄存器地址Rn1和Rn2，和一个5位被写寄存器地址Wn，同时还包括一个1位的写使能信号Write，一个32位的被写数据输入端口Wd及一个时钟信号端口。输出包括两个32位的读寄存器结果A和B。在读数据时，该寄存器堆相当于组合逻辑器件，可以支持双口输出；在写数据时，该寄存器堆为时序逻辑器件，且只能单口写入。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：RegFile，然后输入其实现代码：

module RegFile(

input [4:0] Rn1,Rn2,Wn,

//A口输出寄存器编号,B口输出寄存器编号,待写寄存器编号

input Write,//写信号

input [31:0] Wd, //待写数据

output [31:0] A,B,//输出数据

input Clock

);

reg [31:0] Register[1:31]; //定义31个32位的寄存器

//双口读寄存器（0号寄存器内容始终为0）

assign A = (Rn1 == 0)? 0 : Register[Rn1];

assign B = (Rn2 == 0)? 0 : Register[Rn2];

//写寄存器

always @ ( posedge Clock) begin

if (( Write ) && ( Wn != 0)) Register[Wn] <= Wd;

end

integer i;

initial begin

//初始化寄存器内容

for ( i = 1 ; i <= 31 ; i = i + 1) Register [i] = i ;

end

endmodule

代码8-17 模块RegFile的实现

在ISE集成开发环境中，对模块RegFile进行综合，综合结果如图8-17所示。

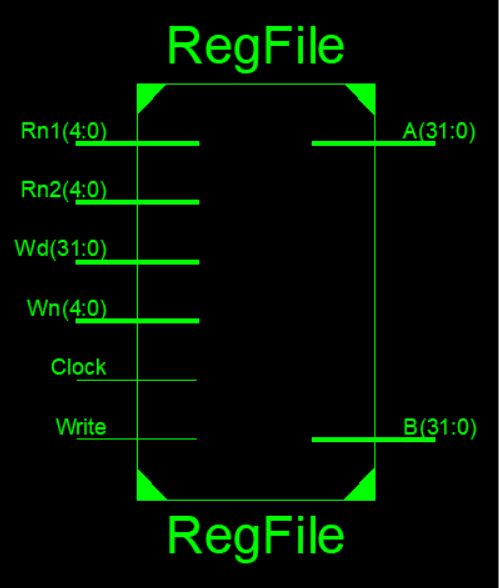


图8-17 模块RegFile的综合结果

在ISE集成开发环境中，对模块RegFile进行仿真。首先输入如下测试代码：

initial begin

Rn1 = 0;

Rn2 = 1;

Wn = 0;

Write = 0;

Wd = 0;

Clock = 0;

#100;

#100;

Rn1 = 0;

Rn2 = 1;

Wn = 2;

Write = 1;

Wd = 32'hffff0000;

#100;

#100;

Rn1 = 0;

Rn2 = 1;

Wn = 3;

Write = 1;

Wd = 32'h0000ffff;

#100;

#100;

Rn1 = 2;

Rn2 = 3;

Wn = 0;

Write = 0;

Wd = 0;

#100;

#100;

end

always begin

#100;Clock=~Clock;

end

代码8-18 模块RegFile的激励

然后进行仿真，仿真结果如图8-18所示。在0~200ns时Write=0，Rn1=0，Rn2=1，A=0，B=1（寄存器有初始值）；在200~400ns时，Write=1，Wn=2，Wd=0xffff0000（写0xffff0000到2号寄存器）；在400~600ns时，Write=1，Wn=3，Wd=0x0000ffff（写0x0000ffff到3号寄存器）；在600~800ns时，Write=0，Rn1=2，Rn2=3，A=0xffff0000，B=0x0000ffff（读2号和3号寄存器）。

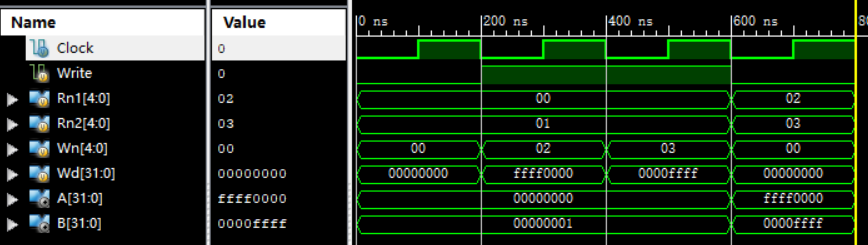


图8-18 模块RegFile的仿真结果

1. 控制器（Control Unit）设计与实现

控制器的功能为分析指令并为各部件协调工作提供相应的控制信号，可将其大体分为两个主要的部分：主控单元Control和ALU局部控制单元ALUOp。本实验沿用模块化的思想，接口制定完成后，先分别实现主控单元和ALU局部控制单元，再将两者通过实例化语句组装成完整的控制器。

1. 主控制单元的设计与实现

主控单元的输入为6位操作码OP，输出为目的寄存器选择信号RegDst，寄存器写使能信号RegWrite，ALU操作数选择信号ALUSrc，数据存储器写使能信号MemWrite, 数据存储器读使能信号MemRead，待写寄存器数据选择信号MemtoReg，条件跳转控制信号Branch，无条件跳转控制信号Jump和输入到ALU局部控制单元的2位分类信号ALUctr，其不同OP对应的输出如下表：

表8-3 主控单元的输入输出

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 输入 | | 输出 | | | | | | | | |
| RegDst | RegWrite | ALUSrc | MemWrite | MemRead | MemtoReg | Branch | Jump | ALUctr[1:0] |
| 指令 | OP[5:0] |
| R-Type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 00 |
| sw | 101011 | x | 0 | 1 | 1 | 0 | x | 0 | 0 | 00 |
| beq | 000100 | x | 0 | 0 | 0 | 0 | x | 1 | 0 | 01 |
| lui | 001111 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 11 |
| j | 000010 | x | 0 | x | 0 | 0 | x | 0 | 1 | xx |

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Control，然后输入其实现代码：

module Control(

input [5:0] op,//操作码

output RegDst , RegWrite , ALUSrc ,

output MemWrite,MemRead , MemtoReg ,

output Branch , Jump,

output [1:0] ALUctr

//传递给ALU局部控制单元的分类信息

);

//指令译码电路

wire i\_Rt = ~|op;//000000

wire i\_lw = op[5] & ~op[3];//100011

wire i\_sw = op[5] & op[3];//101011

wire i\_beq = op[2] & ~op[1];//000100

wire i\_lui = op[3] & op[2];//001111

wire i\_j = op[1] & ~op[0];//000010

//信号生成电路

assign RegDst = i\_Rt;

assign RegWrite = i\_Rt | i\_lw | i\_lui;

assign ALUSrc = i\_lw | i\_sw | i\_lui;

assign MemWrite = i\_sw;

assign MemRead = i\_lw;

assign MemtoReg = i\_lw;

assign Branch = i\_beq;

assign Jump = i\_j;

assign ALUctr[1] = i\_Rt | i\_lui;

assign ALUctr[0] = i\_beq | i\_lui;

//00 - lw, sw; 01 - beq; 10 - R-type; 11 - lui

endmodule

代码8-19 模块Control的实现

在ISE集成开发环境中，对模块Control进行综合，综合结果如图8-19所示。

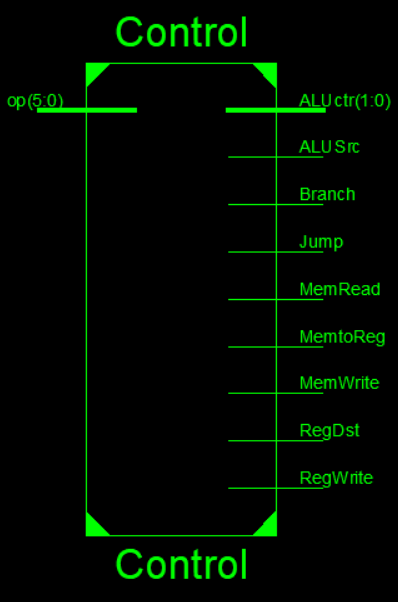


图8-19 模块Control的综合结果

在ISE集成开发环境中，对模块Control进行仿真。首先输入如下测试代码：

initial begin

op = 6'b000000; //R-type

#100;

op = 6'b100011; //lw

#100;

op = 6'b101011; //sw

#100;

op = 6'b000100; //beq

#100;

op = 6'b001111; //lui

#100;

op = 6'b000010; //j

end

代码8-20 模块Control的激励

然后进行仿真，仿真结果如图8-20、表8-4所示。

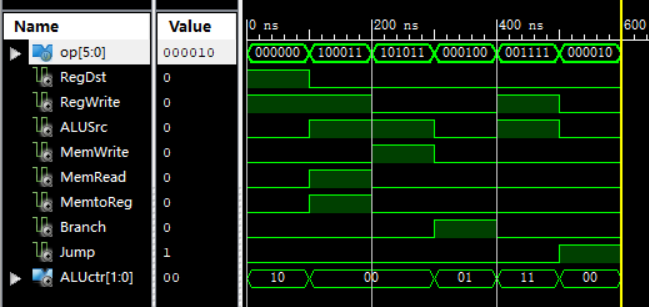


图8-20 模块Control的仿真结果

表8-4 模块Control的仿真结果统计

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OP | RegDst | RegWrite | ALUSrc | MemWrite | MemRead | MemtoReg | Branch | Jump | ALUCtr |
| 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| 100011 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 00 |
| 101011 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 00 |
| 000100 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 01 |
| 001111 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 11 |
| 000010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 |

1. ALU局部控制单元的设计与实现

ALU局部控制单元的输入为6位功能码func和来自主控单元的2位分类码ALUctr；输出为算数逻辑单元的控制信号ALU\_op，控制ALU具体进行什么运算。其输入输出表如下：

表8-5 ALU局部控制单元输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入 | | | | 输出 |
| 指令 | 功能 | func[5:0] | ALUctr[1:0] | ALU\_op[2:0] |
| R-Type | add | 100000 | 10 | 000 |
| sub | 100010 | 100 |
| and | 100100 | 001 |
| or | 100101 | 101 |
| xor | 100110 | 010 |
| lw | 取数 | xxxxxx | 00 | 000 |
| sw | 存数 |
| beq | 分支 | 01 | 100 |
| lui | 置高位 | 11 | 110 |
| j | 跳转 | xx | xxx |

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：ALUop，然后输入其实现代码：

module ALUop(

input [5:0] func, //功能码

input [1:0] ALUctr, //ALU局部控制单元的分类输入

output [2:0] ALU\_op //ALU控制信号

);

wire i\_Rt = ALUctr[1] & ~ALUctr[0]; //Rt - 10

assign ALU\_op[2] = (i\_Rt & ((~func[2] & func[1]) | (func[2] & func[0]))) | ALUctr[0];

// sub + or + beq + lui

assign ALU\_op[1] = (i\_Rt & func[2] & func[1]) | (ALUctr[1] & ALUctr[0]);

// xor + lui

assign ALU\_op[0] = (i\_Rt & func[2] & ~func[1]);

// and + or

endmodule

代码8-21 模块ALUop的实现

在ISE集成开发环境中，对模块ALUop进行综合，综合结果如图8-21所示。

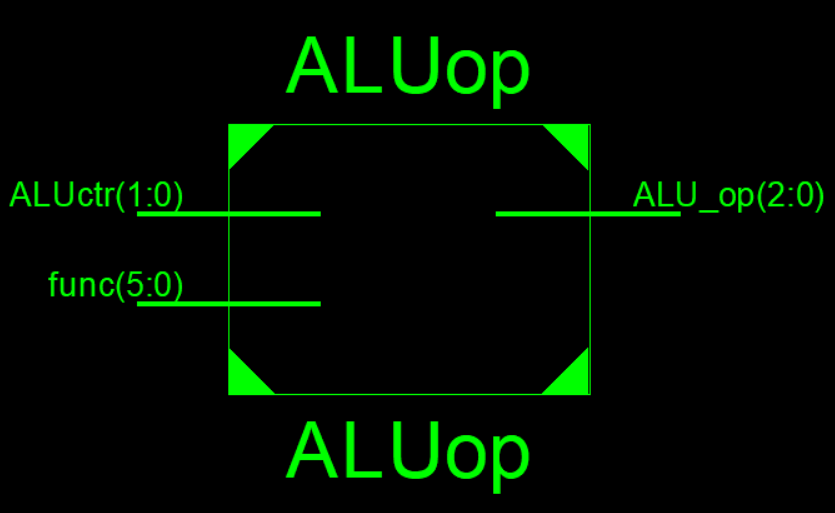


图8-21 模块ALUop的综合结果

在ISE集成开发环境中，对模块ALUop进行仿真。首先输入如下测试代码：

initial begin

ALUctr = 2'b10;

func = 6'b100000;

#100;

func = 6'b100010;

#100;

func = 6'b100100;

#100;

func = 6'b100101;

#100;

func = 6'b100110;

#100;

ALUctr = 2'b00;

#100;

ALUctr = 2'b01;

#100;

ALUctr = 2'b11;

end

代码8-22 模块ALUop的激励

然后进行仿真，仿真结果如图8-22、表8-6所示。

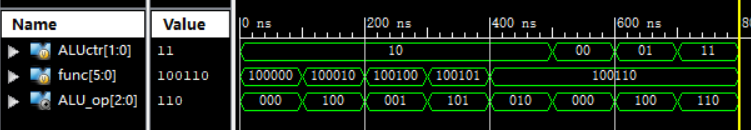


图8-22 模块ALUop的仿真结果

表8-6 模块ALUop的仿真结果统计

|  |  |  |
| --- | --- | --- |
| ALUCtr | func | ALU\_op |
| 10 | 100000 | 000 |
| 100010 | 100 |
| 100100 | 001 |
| 100101 | 101 |
| 100110 | 010 |
| 00 | xxxxxx | 000 |
| 01 | xxxxxx | 100 |
| 11 | xxxxxx | 110 |

1. 控制器的设计与实现

如上文所述，控制器为利用实例化语法将前面两个模块组装而成。其输入为操作码OP和功能码Func，输出为各种控制信号。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Control\_Unit，然后输入其实现代码：

module Control\_Unit(

input [5:0] op , func,

//操作码,功能码

output RegDst, RegWrite, ALUSrc,

output MemWrite, MemRead, MemtoReg,

output Branch, Jump,

output [2:0] ALU\_op //ALU控制信号

);

wire [1:0] ALUctr;//传递给ALU局部控制单元的分类信息

Control mainControl( .op(op),

.RegDst(RegDst) ,

.RegWrite(RegWrite) ,

.ALUSrc(ALUSrc) ,

.MemWrite(MemWrite),

.MemRead(MemRead) ,

.MemtoReg(MemtoReg) ,

.Branch(Branch) ,

.ALUctr(ALUctr),

.Jump(Jump));

ALUop aluControl(.func(func),.ALUctr(ALUctr), .ALU\_op(ALU\_op));

endmodule

代码8-23 模块Control\_Unit的实现

在ISE集成开发环境中，对模块Control\_Unit进行综合，综合结果如图8-23所示。

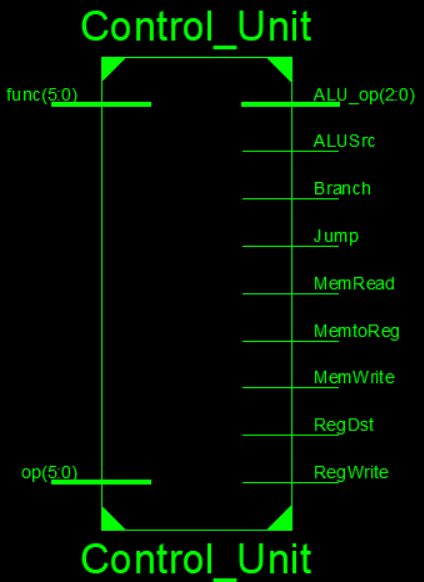


图8-23 模块Control\_Unit的综合结果

在ISE集成开发环境中，对模块Control\_Unit进行仿真。首先输入如下测试代码：

initial begin

op = 6'b000000; //R-type

func=6'b100000; //Add

#100;

func=6'b100010; //Sub

#100;

func=6'b100100; //And

#100;

func=6'b100101; //Or

#100;

func=6'b100110; //Xor

#100;

op = 6'b100011; //lw

#100;

op = 6'b101011; //sw

#100;

op = 6'b000100; //beq

#100;

op = 6'b001111; //lui

#100;

op = 6'b000010; //j

end

代码8-24 模块Control\_Unit的激励

然后进行仿真，仿真结果如图8-24所示。

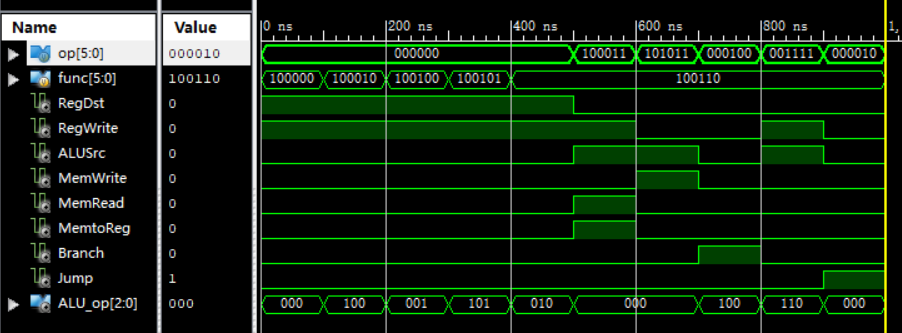


图8-24 模块Control\_Unit的仿真结果

表8-7 模块Control\_Unit的仿真结果统计

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| op | 000000 | 000000 | 000000 | 000000 | 000000 | 100011 | 101011 | 000100 | 001111 | 000010 |
| func | 100000 | 100010 | 100100 | 100101 | 100110 | 100110 | 100110 | 100110 | 100110 | 100110 |
| regdst | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| regwrite | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| alusrcr | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| memwrite | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| memread | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| memtoreg | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| branch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| jump | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| alu\_op | 000 | 100 | 001 | 101 | 010 | 000 | 000 | 100 | 110 | 000 |

1. 30位取指部件（Fetch）的设计与实现

取指部件的功能为：在时钟信号的作用下，自动获取下一条指令地址，根据该地址CPU可以按序找下一条要执行的指令。其有7个输入：时钟信号Clock，同步清零信号Reset，条件跳转信号B，30位PC相对寻址偏移量B\_addr，零标志位Z，无条件跳转信号Jump，26位页面寻址的页面偏移Offset。其有一个输出：32位的PC输出值。

PC的更新逻辑可归纳如下：

1. 默认不发生跳转，PC=PC+4
2. 当B=1并且Z=1时条件跳转，PC=PC+4+B\_addr
3. 当J=1时无条件跳转，PC={PC高4位，Offset}

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：Fetch\_30，然后输入其实现代码：

module Fetch\_30(

input B , Z , J, Reset , Clock ,

//Branch,Zero,Jump,Reset,Clock信号

input [29:0] B\_addr, //分支指令的相对偏移

input [25:0] Offset, //J指令的页面偏移

output [31:0] addr //新的PC

);

reg [29:0] PC; //内部使用30位地址

assign addr = {PC,2'b00}; //外部输出32位地址

wire [29:0] sum0 , sum1 , splicePC, branchPC, nextPC;

wire beqSel = Z & B; //Zero && Branch

ADD30 add1( PC , 1 , sum0); //PC=PC+1

ADD30 add2( sum0, B\_addr , sum1); //PC=PC+1+B\_addr

assign splicePC={PC[29:26],Offset};

MUX30\_2\_1 muxBranch(sum0 , sum1 , beqSel , branchPC);

MUX30\_2\_1 muxJump(branchPC,splicePC,J,nextPC);

always @(posedge Clock ) begin

if (Reset == 1) PC = 0; //清零

else PC = nextPC;

end

initial PC = 0; //初始化PC

endmodule

代码8-25 模块Fetch\_30的实现

在ISE集成开发环境中，对模块Fetch\_30进行综合，综合结果如图8-25所示。

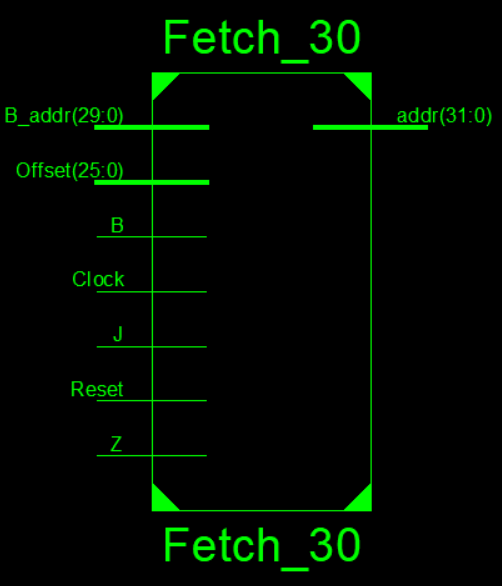


图8-25 模块Fetch\_30的综合结果

在ISE集成开发环境中，对模块Fetch\_30进行仿真。首先输入如下测试代码：

initial begin

B = 0;

Z = 0;

J = 0;

Reset = 0;

Clock = 0;

B\_addr = 0;

Offset = 0;

#90;

Reset = 1;

#20;

Reset = 0;

#180;

B = 1;

Z = 1;

B\_addr = 30'h3fffc000;

#10;

#100;

B = 0;

Z = 0;

B\_addr = 30'b0;

#100;

#100;

#100;

#100;

J = 1;

Offset = 26'b1;

end

always begin

#100;Clock=~Clock;

end

代码8-26 模块Fetch\_30的激励

然后进行仿真，仿真结果如图8-26所示。由波形图可见，时钟信号周期为200ns。在第一个上升沿，Reset=1，addr=0（同步清零）；在第二个上升沿，B=1，Z=1，B\_addr=0x3fffc000，addr=0xffff0004（条件跳转）；第三个上升沿，B=0，J=0，Reset=0，addr=0xffff0008（自增）；第四个上升沿，B=0，J=0，Reset=0，addr=0xffff000c（自增）；第五个上升沿，J=1，Offset=1，addr=0xf0000004（无条件跳转）。

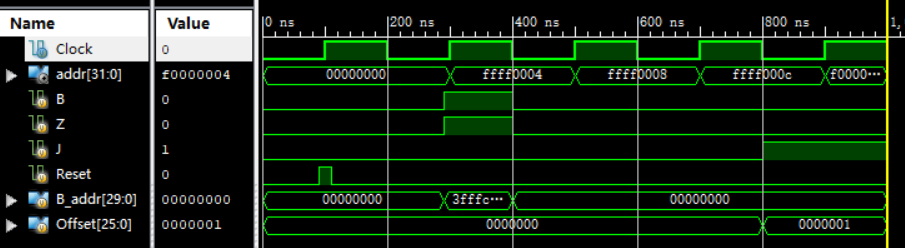


图8-26 模块Fetch\_30的仿真结果

1. 数据通路（Data\_Flow）的设计与实现（CPU的封装）

CPU封装的基本原理就是根据单周期CPU的总体电路将各功能部件按照指令的数据通路连接起来。

在ISE集成开发环境中，在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，创建一个Verilog Module模块，名称为：DATA\_FLOW，然后输入其实现代码：

module DATA\_FLOW(

input [31:0] Inst,

input Reset, Clock,

input [31:0] data,

output [31:0] Aluout, B, PC,

output MemWrite, MemRead, MemtoReg

);

wire RegDst, RegWrite, ALUSrc, Branch, Zero, Jump;

wire [2:0] ALU\_op; wire [4:0] Wn;

//通路内部控制信号

wire [31:0] A, ALU\_B, Ext\_Imm; wire [29:0] Ext\_Imm\_30;

//寄存器A口输出,ALU单元B口输入,32符号扩展的立即数,30位符号扩展的立即数

Sign\_Extender\_30 ext30( Inst[15:0] , Ext\_Imm\_30);

Sign\_Extender ext32( Inst[15:0] , Ext\_Imm);

//由于目标指令集不包含I型指令的逻辑运算,只需符号扩展

MUX5\_2\_1 muxRegDst( Inst[20:16], Inst[15:11], RegDst, Wn );

MUX32\_2\_1 muxAluSrc( B, Ext\_Imm, ALUSrc, ALU\_B );

Fetch\_30 fetch30(Branch, Zero, Jump, Reset, Clock, Ext\_Imm\_30, Inst[25:0], PC);

//内部30位的取指单元,无需移位器件

Control\_Unit control( .op(Inst[31:26]) , .func(Inst[5:0]),

.RegDst(RegDst) , .RegWrite(RegWrite) , .ALUSrc(ALUSrc) ,

.MemWrite(MemWrite),.MemRead(MemRead) , .MemtoReg(MemtoReg) ,

.Branch(Branch) , .Jump(Jump), .ALU\_op(ALU\_op) );

//控制单元,根据操作码和功能码产生控制信号

RegFile regFile(Inst[25:21], Inst[20:16], Wn, RegWrite, data, A, B, Clock);

//寄存器堆,双口读,单口同步写

ALU alu( A, ALU\_B, ALU\_op, Aluout, Zero);

//ALU,根据AB口输入和控制信号产生运算结果和状态码

endmodule

代码8-27 模块DATA\_FLOW的实现

在ISE集成开发环境中，对模块DATA\_FLOW进行综合，综合结果如图8-27所示。

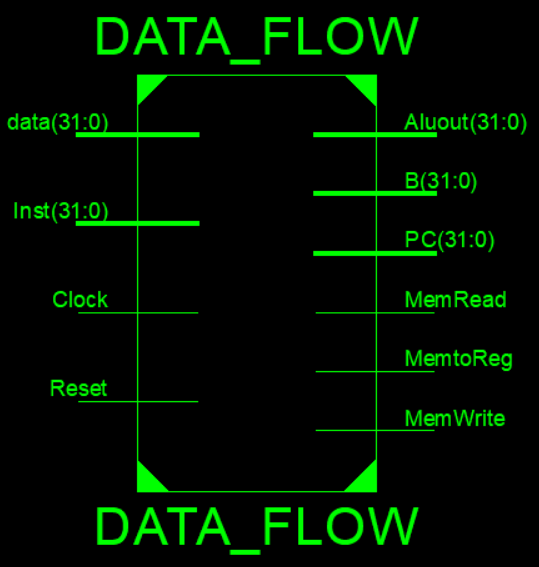


图8-27 模块DATA\_FLOW的综合结果

在ISE集成开发环境中，对模块DATA\_FLOW进行仿真。首先输入如下测试代码：

initial begin

Reset = 0;

Clock = 0;

//lw R1,R0,offset;

Inst = 32'b100011\_00000\_00001\_00000\_00000\_000000;

data = 1;

#100;

#100;

//lw R2,R0,offset;

Inst = 32'b100011\_00000\_00010\_00000\_00000\_000000;

data = 2;

#100;

#100;

//add R3,R1,R2;

Inst = 32'b000000\_00001\_00010\_00011\_00000\_100000;

data = 0;

#100;

#100;

end

always begin

#100;Clock=~Clock;

end

代码8-28 模块DATA\_FLOW的激励

然后进行仿真，仿真结果如图8-28所示。如波形图所示，时钟信号周期为200ns，在第一个上升沿，Inst=0x8c010000，data=1，MemRead=1，MemtoReg=1（lw R1,R0,offset）；在第二个上升沿，Inst=0x8c020000，data=2，MemRead=1，MemtoReg=1（lw R2,R0,offset）；在第三个上升沿，Inst=0x00201820，MemWrite=0，MemRead=0，MemtoReg=0，Aluout=3（add R3,R1,R2）。

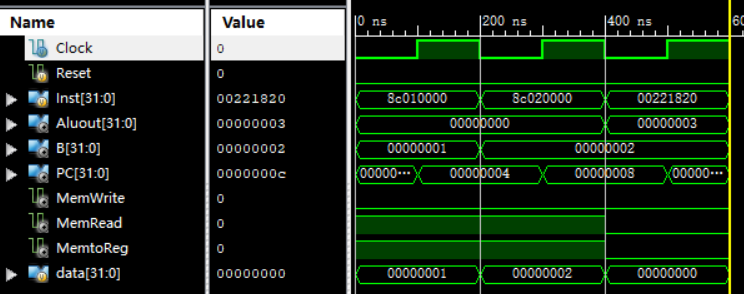


图8-28 模块DATA\_FLOW的仿真结果

1. 测试
2. 添加指令存储器

根据冯诺依曼存储程序的思想，该系统需要一个指令存储器。出于测试的简洁性，选择ROM作为指令存储器。ROM的具体内容如表8-8所示：

表8-8 ROM各空间对应指令

|  |  |  |
| --- | --- | --- |
| 地址 | 指令 | 操作 |
| 0x0 | Add $a1, $0, $0 | 置$a1为0 |
| 0x4 | Lw $s1, 0($a1) | 读Mem[0]到$s1 |
| 0x8 | Lw $s2, 4($a1) | 读Mem[4]到$s2 |
| 0xc | Sub $s3,$s1,$s2 | 置$s3为$s1-$s2 |
| 0x10 | Add $s1,$s1,$s2 | 置$s1为$s1+$s2 |
| 0x14 | Lw $s2, 8($a1) | 读Mem[8]到$s2 |
| 0x18 | Beq $s1, $s2, 2 | 根据$s1和$s2进行条件跳转 |
| 0x1c | Sw $s3, 12($a1) | $s3到Mem[12] |
| 0x20 | J 0 | 无条件跳转，构成循环 |
| 0x24 | Sw $s1, 12($a1) | 写$s1到Mem[12] |

在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，选择Verilog Module输入，并输入Verilog文件名：Inst\_rom，单击Next按钮进入端口定义对话框。其中Module Name栏输入模块名Inst\_rom，单击Next进入下一步，点击Finish完成创建。实现Inst\_rom的Verilog程序参考代码：

module Inst\_rom(

input [31:0] address, //PC

output [31:0] inst

);

wire [31:0] rom [0:31]; //最多32条指令,可用5bit编码地址

assign rom[5'h00]=32'h00002820; //Add $a1, $0, $0

assign rom[5'h01]=32'h8Cb10000; //Lw $s1, 0($a1)

assign rom[5'h02]=32'h8cb20004; //Lw $s2, 4($a1)

assign rom[5'h03]=32'h02329822; //Sub $s3,$s1,$s2

assign rom[5'h04]=32'h02328820; //Add $s1,$s1,$s2

assign rom[5'h05]=32'h8cb20008; //Lw $s2, 8($a1)

assign rom[5'h06]=32'h12320002; //Beq $s1, $s2, 2

assign rom[5'h07]=32'haCb3000c; //Sw $s3, 12($a1)

assign rom[5'h08]=32'h08000000; //J 0

assign rom[5'h09]=32'hacb1000c; //Sw $s1, 12($a1)

assign inst=rom[address[6:2]];

endmodule

代码8-29 模块Inst\_rom的实现

在ISE集成开发环境中，对模块Inst\_rom进行综合，综合结果如图8-29所示。

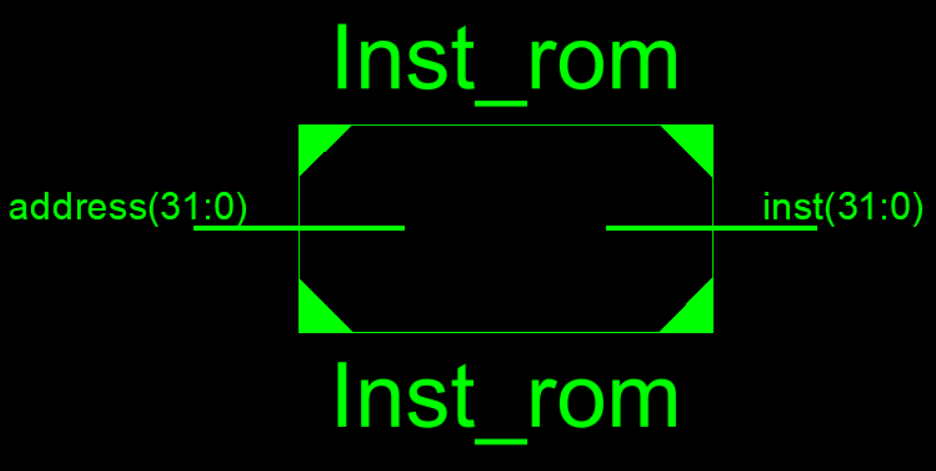


图8-29 模块Inst\_rom的综合结果

在ISE集成开发环境中，对模块Inst\_rom进行仿真。首先输入如下测试代码：

integer i;

initial begin

address = 0;

for ( i = 1 ; i <= 9 ; i = i + 1) begin

#50;

address = address + 4;

end

end

代码8-30 模块Inst\_rom的激励

然后进行仿真，仿真结果如图8-30所示。

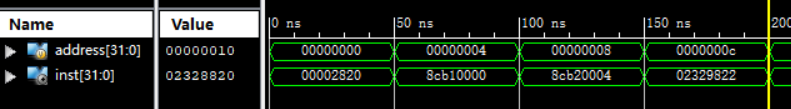


图8-30 模块Inst\_rom的仿真结果

1. 添加数据存储器

由于系统中寄存器的数量较少，成本较高，系统需要添加数据存储器作为内存以解决数据存储问题。

在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，选择Verilog Module输入，并输入Verilog文件名：Data\_ram，单击Next按钮进入端口定义对话框。其中Module Name栏输入模块名Data\_ram，单击Next进入下一步，点击Finish完成创建。实现Data\_ram的Verilog程序参考代码：

module Data\_ram(

input Clock, //同步时序器件,上升沿触发

input [31:0] datain,//待写数据

input [31:0] addr,//内存地址,按字节编制

input write,read, //读写信号

output [31:0] dataout//读出数据

);

reg [31:0] ram[0:31]; //最多32个字

assign dataout = read ? ram[addr[6:2]] : 32'hxxxxxxxx;

always @ (posedge Clock) begin

if ( write == 1) ram[addr[6:2]]=datain;

end

initial begin

//初始化内存单元内容

ram[5'h00]=32'h2;

ram[5'h01]=32'h3;

ram[5'h02]=32'h6;

end

endmodule

代码8-31 模块Data\_ram的实现

在ISE集成开发环境中，对模块Data\_ram进行综合，综合结果如图8-31所示。

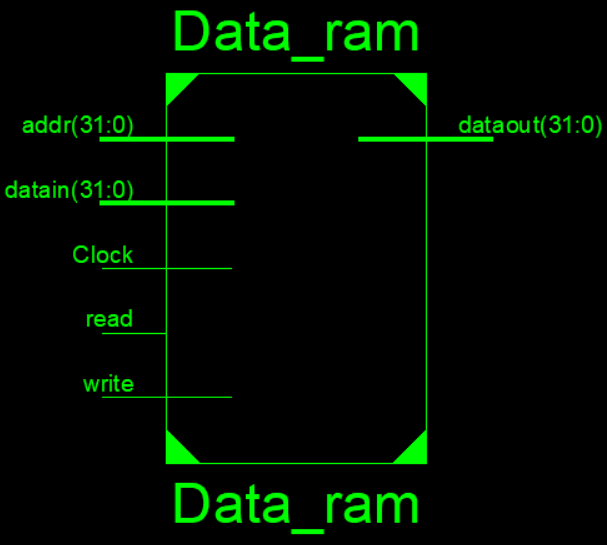


图8-31 模块Data\_ram的综合结果

在ISE集成开发环境中，对模块Data\_ram进行仿真。首先输入如下测试代码：

initial begin

Clock = 0;

datain = 0;

addr = 0; //pos 0 val 2

write = 0;

read = 1;

#100;

#100;

addr = 32'h4;//pos 1 val 3

write = 0;

read = 1;

#100;

#100;

addr = 32'h8;

datain = 32'hffffffff;

write = 1;

read = 0;

#100;

#100;

addr = 32'h8;

datain = 0;

write = 0;

read = 1;

#100;

#100;

end

always begin

#100;Clock=~Clock;

end

代码8-32 模块Data\_ram的激励

然后进行仿真，仿真结果如图8-32所示。

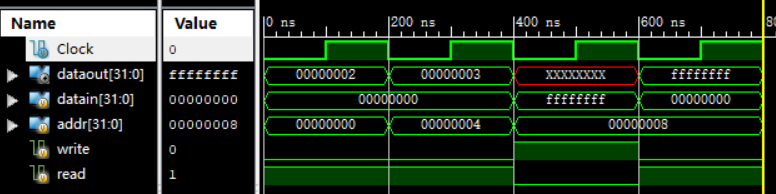


图8-32 模块Data\_ram的仿真结果

1. 仿真测试

仿真测试之前，首先需要封装主板，即为将CPU、指令存储器和数据存储器连接起来，使其成为受时钟信号控制的一个统一整体。

在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择New Source命令，选择Verilog Module输入，并输入Verilog文件名：MainBoard，单击Next按钮进入端口定义对话框。其中Module Name栏输入模块名MainBoard，单击Next进入下一步，点击Finish完成创建。实现MainBoard的Verilog程序参考代码：

module MainBoard(

input Reset , Clock, //PC清零信号和去抖动后的单步脉冲

output [31:0] Inst,PC,Aluout,B\_data //系统输出，供调试使用

);

wire MemWrite, MemRead,MemtoReg;

//内存写信号,内存读信号,寄存器待写数据源选择信号

wire [31:0] MemOutData , data;

//内存输出数据,寄存器待写数据

DATA\_FLOW path( Inst, Reset , Clock, data, Aluout , B\_data , PC , MemWrite, MemRead , MemtoReg );

//将指令内容,清零信号,脉冲信号和数据存储器输出作为输入

//得到ALU运算结果,busB数据,新的PC,内存读写信号,和寄存器数据源控制信号

Inst\_rom rom( PC, Inst );//从指令存储器读指令

Data\_ram ram( Clock, B\_data, Aluout , MemWrite, MemRead, MemOutData);

//将寄存器组busB上的数据作为内存待写数据,ALU运算结果作为内存地址

//并根据读写信号得到内存输出

MUX32\_2\_1 mux(Aluout , MemOutData , MemtoReg , data);

//选择ALU运算结果或内存输出数据作为寄存器待写数据

endmodule

代码8-33 模块MainBoard的实现

在ISE集成开发环境中，对模块MainBoard进行综合，综合结果如图8-33所示。

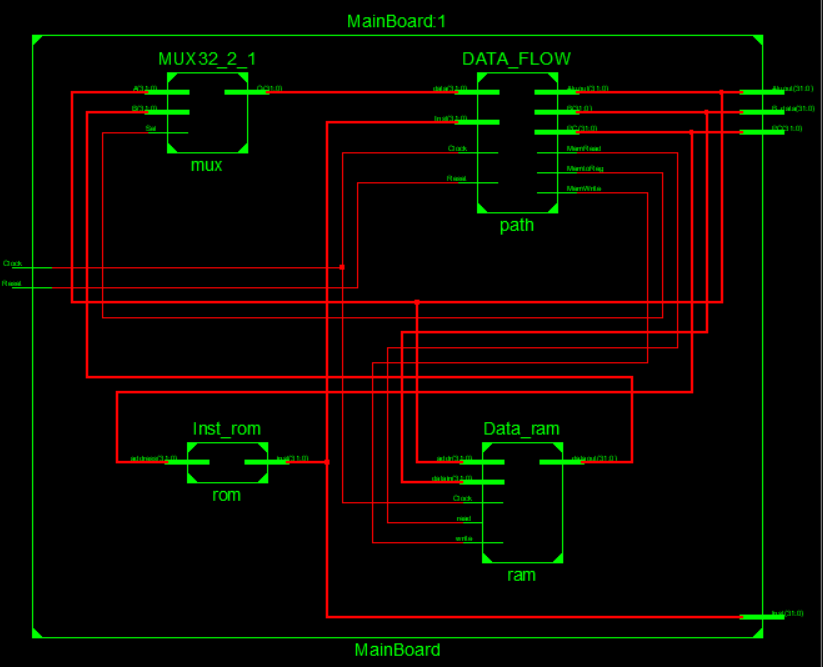


图8-33 模块MainBoard的综合结果

在ISE集成开发环境中，对模块MainBoard进行仿真。首先输入如下测试代码：

initial begin

Reset = 0;

Clock = 0;

#100;

#100;

Reset = 1;

#100

#100

Reset = 0;

end

always begin

#100;Clock=~Clock;

end

代码8-34 模块MainBoard的激励

然后进行仿真，仿真结果如图8-34所示。

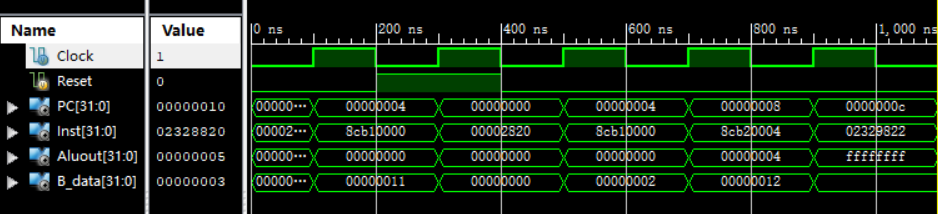


图8-34(a) 模块MainBoard的仿真结果(0~1000ns)

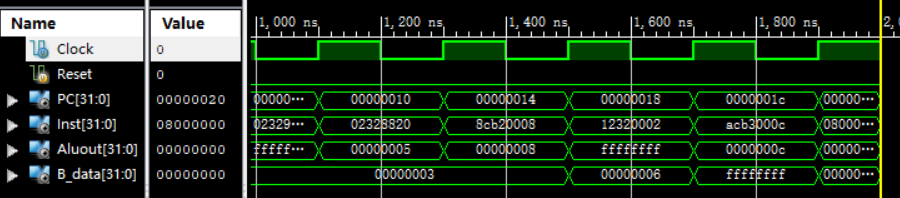


图8-34(b) 模块MainBoard的仿真结果(1000~2000ns)

1. 板级测试

仿真测试通过后，下一步要做的工作就是将所做设计下载到开发板进行验证。为了便于测试和观察，对开发板上的器件作如下约定：

* 按钮Button[0]，作测试时钟脉冲输入
* 按钮Button[1]，作为PC同步置零信号
* 开关Switch[1:0]：输出内容选择，00 – 显示执行指令，01 – 显示PC，10 – 显示ALU运算结果，11 – 显示数据存储器输入
* 开关Switch[2]：0 – 显示输出内容的低24位，1 – 显示输出内容的高24位
* 七段数码管组（共六个）显示内容

由于通过按键模拟时钟脉冲会产生毛刺，故Button[0]的输入信号需要首先通过按键去抖电路以消除毛刺（毛刺可被误判为脉冲）。去抖电路实现如下：

module BTN\_Anti\_Jitter(

input Clock,

input BTN\_IN, //输入按键状态

output reg BTN\_Out //输出按键状态

);

reg [3:0] cnt;

//按键按下和抬起时可能产生毛刺

//去除毛刺的思路就是让维持时间短于阈值的变化都视为毛刺并忽略

reg BTN\_Old; //上次按键状态

always @ (posedge Clock) begin

if(BTN\_IN != BTN\_Old)

begin

cnt <= 4'b0000; BTN\_Old <= BTN\_IN;

//计数器清零并更新上次按键状态

end

else

begin

if( cnt == 4'b1111)

begin

cnt <= 4'b0000; BTN\_Out <= BTN\_IN;

//计数器清零并更新输出按键状态

end

else

cnt <= cnt + 1'b1;

//计数器+1

end

end

endmodule

代码8-35 模块BTN\_Anti\_Jitter的实现

同时由于需要将主板内信息通过七段数码管组进行输出，故需要七段数码管组驱动模块。驱动电路实现如下：

module Hex7seg\_decode(

input wire [23:0] disp\_num , //数码管组输入内容

input wire[2:0] Scanning, //扫描信号

output reg [7:0] digit\_seg, //7段数码管输入编码

output reg [5:0] AN\_SEL //数码管选择信号

);

reg [3:0] digit;

//一个十六进制数

always @ (\*) begin

AN\_SEL = 6'b000000;//默认不做修改

case (Scanning)

//根据扫描信号选择对应的十六进制数和对应的数码管选择信号

3'h0: begin digit[3:0] = disp\_num[3:0]; AN\_SEL = 6'b000001; end

3'h1: begin digit[3:0] = disp\_num[7:4]; AN\_SEL = 6'b100000; end

3'h2: begin digit[3:0] = disp\_num[11:8]; AN\_SEL = 6'b010000; end

3'h3: begin digit[3:0] = disp\_num[15:12]; AN\_SEL = 6'b001000; end

3'h4: begin digit[3:0] = disp\_num[19:16]; AN\_SEL = 6'b000100; end

3'h5: begin digit[3:0] = disp\_num[23:20]; AN\_SEL = 6'b000010; end

endcase

end

always @ (\*) begin

case (digit)

//将一个十六进制数转换为7段数码管输入编码

4'h0: digit\_seg = 8'b00111111;

4'h1: digit\_seg = 8'b00000110;

4'h2: digit\_seg = 8'b01011011;

4'h3: digit\_seg = 8'b01001111;

4'h4: digit\_seg = 8'b01100110;

4'h5: digit\_seg = 8'b01101101;

4'h6: digit\_seg = 8'b01111101;

4'h7: digit\_seg = 8'b00000111;

4'h8: digit\_seg = 8'b01111111;

4'h9: digit\_seg = 8'b01101111;

4'hA: digit\_seg = 8'b01110111;

4'hB: digit\_seg = 8'b01111100;

4'hC: digit\_seg = 8'b00111001; //C

4'hD: digit\_seg = 8'b01011110; //d

4'hE: digit\_seg = 8'b01111001;

4'hF: digit\_seg = 8'b01110001;

endcase

end

endmodule

代码8-36 模块Hex7seg\_decode的实现

将模块Mainboard的Clock输入连接到按键去抖电路；Reset输入连接到Botton[1]；数据输出经过转换后连接到七段数码管组驱动电路，由此构成单周期计算机。连接代码如下：

module Single\_Cycle\_Computer\_IO(

input Clock, //接收开发板主板时钟输入

input Reset, BTN\_IN, //PC清零信号和单步脉冲输入

input [2:0] SW, //控制七段数码管显示内容

output [7:0] seg, //数码管7段信号

output [5:0] AN\_SEL , //译码后的六个数码管的选择信号

output [31:0] Inst,PC,Aluout,B\_data //系统输出，供调试使用

//output [2:0] LED

);

//assign LED[0] = BTN\_IN;

//assign LED[1] = Reset;

//assign LED[2] = 1;

reg [31:0] clockdiv = 0;//用于分频的寄存器

always @ (posedge Clock) clockdiv <= clockdiv + 1;

//按键去抖动

wire BTN\_OUT;//BTN\_IN去抖动后的按键输出

BTN\_Anti\_Jitter anti\_jitter(clockdiv[12],BTN\_IN,BTN\_OUT);

//数码管显示

wire [23:0] disp; wire [31:0] disp\_tmp;

assign disp\_tmp=(SW[1:0]==2'b00)?Inst:

(SW[1:0]==2'b01)?PC:

(SW[1:0]==2'b10)?Aluout:B\_data;

assign disp = SW[2]?disp\_tmp[31:8]:disp\_tmp[23:0];//数码管将要显示的内容

Hex7seg\_decode hex7(disp , clockdiv[18:16] , seg , AN\_SEL);

//扫描信号是主板时钟分频后的结果

MainBoard mainBoard( Reset , BTN\_OUT , Inst , PC , Aluout , B\_data);

//将PC清零信号和去抖动后的单步脉冲作为主板输入

endmodule

代码8-37 模块Single\_Cycle\_Computer\_IO的实现

在ISE集成开发环境中，对模块Single\_Cycle\_Computer\_IO进行综合，综合结果如图8-35所示。

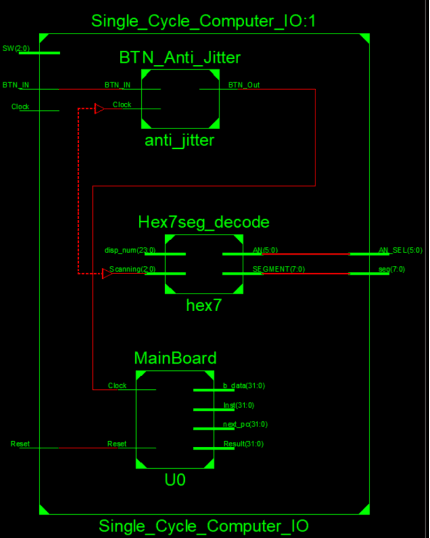


图8-35 模块Single\_Cycle\_Computer\_IO的综合结果

为模块中的输入/输出信号添加管脚约束，即向工程中添加UCF（User Constraints File）文件。在工程管理区单击鼠标右键，点击New Source，在类型中选择Implementation Constraints File，输入文件名：Single\_Cycle\_Computer\_IO。系统会生成一个空白的约束文件并打开。我们就可以为设计添加各种约束。Single\_Cycle\_Computer\_IO.ucf的内容如下：

NET Clock LOC = D11;

Net "Clock" TNM\_NET = sys\_clk\_pin;

#Buttons

Net "BTN\_IN" CLOCK\_DEDICATED\_ROUTE = FALSE;

NET BTN\_IN LOC = E6; //btn(0) clock

NET Reset LOC = D5; //btn(1) reset

#SWs

NET SW(0) LOC=V5;

NET SW(1) LOC=U4;

NET SW(2) LOC=V3;

#Sw[1:0]：00-Inst;01-PC;10-Aluout;11-B\_data

#Sw[2]：0-low 24 bits;1-high 24 bits

#Leds

#NET LED(0) LOC=W3;

#NET LED(1) LOC=Y4;

#NET LED(2) LOC=Y1;

##### 7-seg anode #####

NET AN\_SEL(0) LOC = M17; # 7-seg AN1

NET AN\_SEL(1) LOC = AA20; # 7-seg AN4

NET AN\_SEL(2) LOC = AB21; # 7-seg AN5

NET AN\_SEL(3) LOC = N16; # 7-seg AN2

NET AN\_SEL(4) LOC = P19; # 7-seg AN3

NET AN\_SEL(5) LOC = P16; # 7-seg AN0

##### 7-seg display #####

NET seg(0) LOC=AA21; # 7-seg CA

NET seg(1) LOC=AA22; # 7-seg CB

NET seg(2) LOC=Y22; # 7-seg CC

NET seg(3) LOC=N15; # 7-seg CD

NET seg(4) LOC=AB19; # 7-seg CE

NET seg(5) LOC=P20; # 7-seg CF

NET seg(6) LOC=Y21; # 7-seg CG

NET seg(7) LOC=P15; # 7-seg DP

代码8-38 模块Single\_Cycle\_Computer\_IO的约束文件

约束完成后，下一个步骤就是实现(Implementation)。所谓实现，是指将综合输出的逻辑网表翻译成所选器件的底层模块和硬件原语，将设计映射到器件结构上，进行布局布线，达到在选定器件上实现设计的目的。

在ISE中，执行实现过程，会自动执行翻译、映射和布局布线过程：也可单独执行。在过程管理区双击Implementation Design选项，就可以自动完成实现的3个步骤：Translate、Map、Place & Route，如果没有错，双击Generate Program File，生成Single\_Cycle\_Computer\_IO.bit文件，并将生成的二进制文件下载开发板上验证。

1. **实验数据及结果分析：**

**实验数据：**

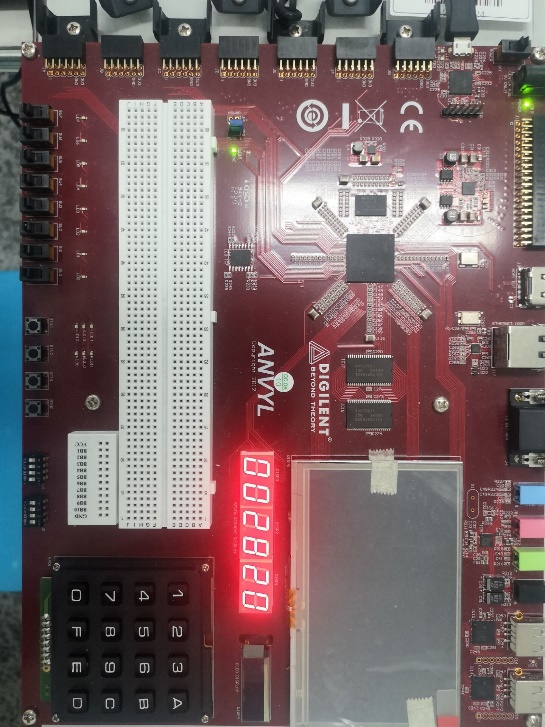
****

图8-36 PC=0，Inst=0x00002820

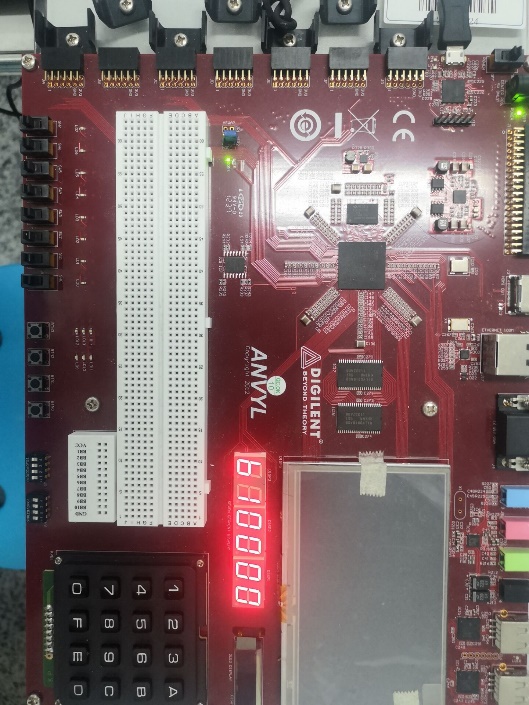


图8-37 PC=4，Inst=0x8Cb10000

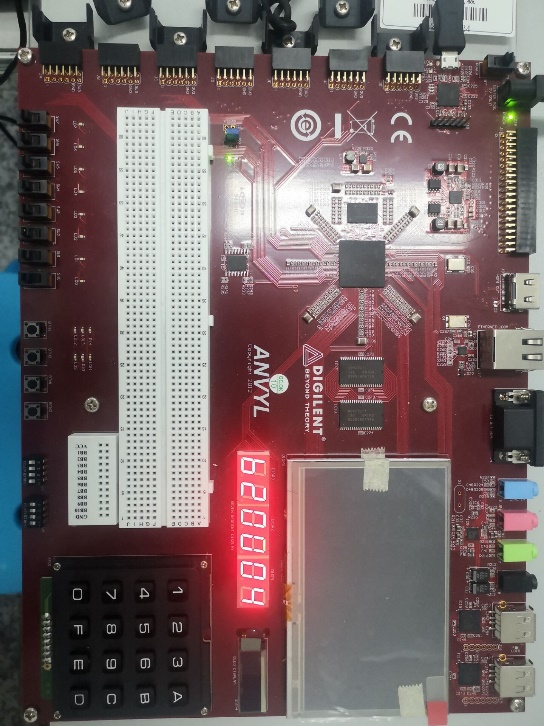


图8-38 PC=8，Inst=0x8cb20004

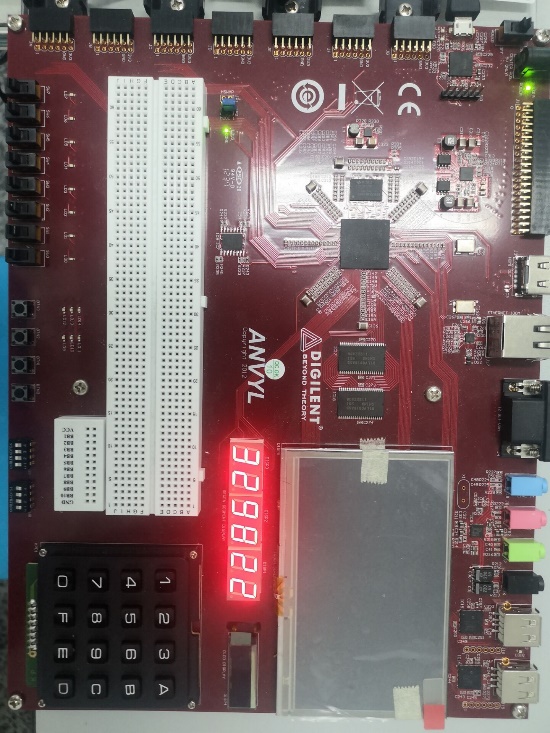


图8-39 PC=12，Inst=0x02329822

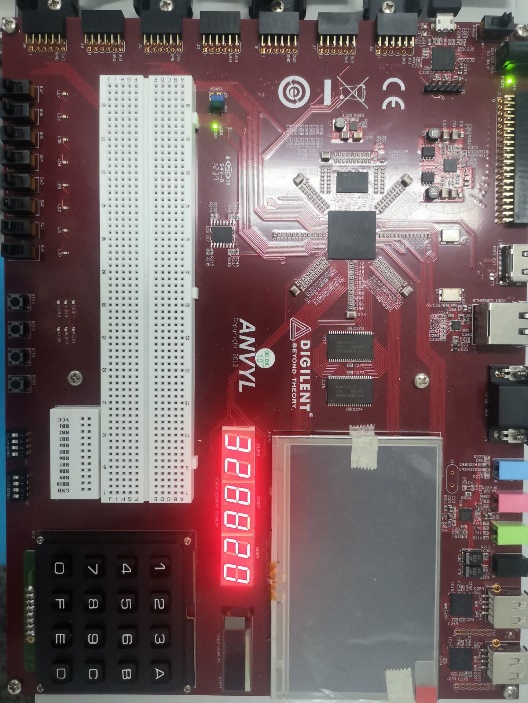


图8-40 PC=16，Inst=0x02328820

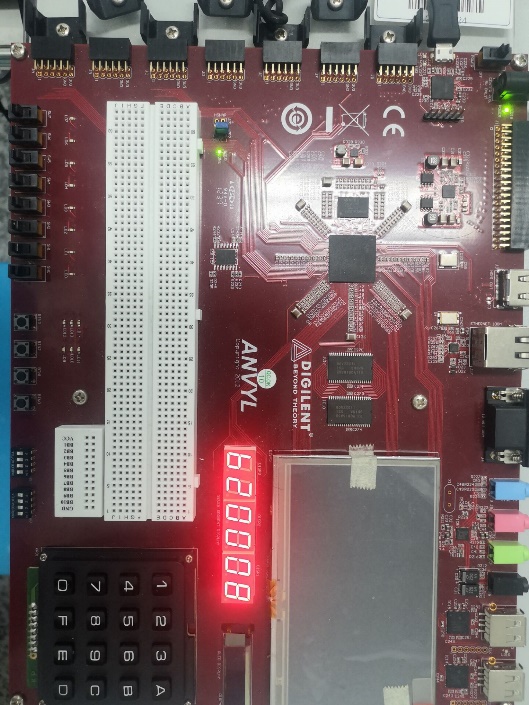


图8-41 PC=20，Inst=0x8cb20008

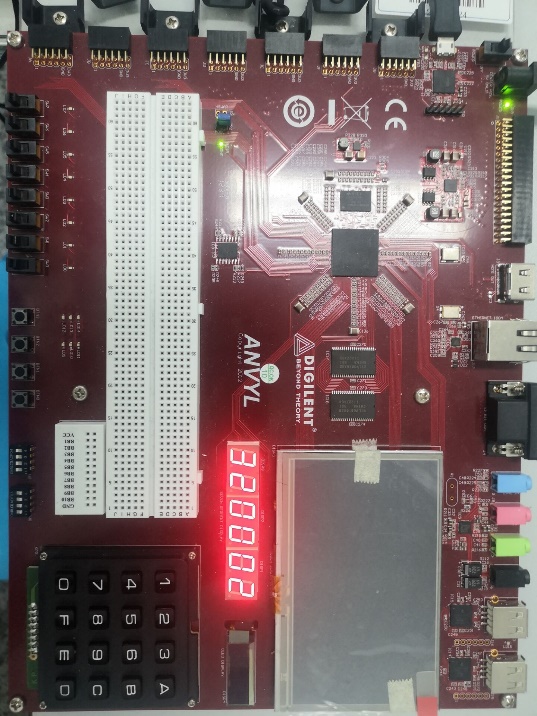


图8-42 PC=24，Inst=0x12320002

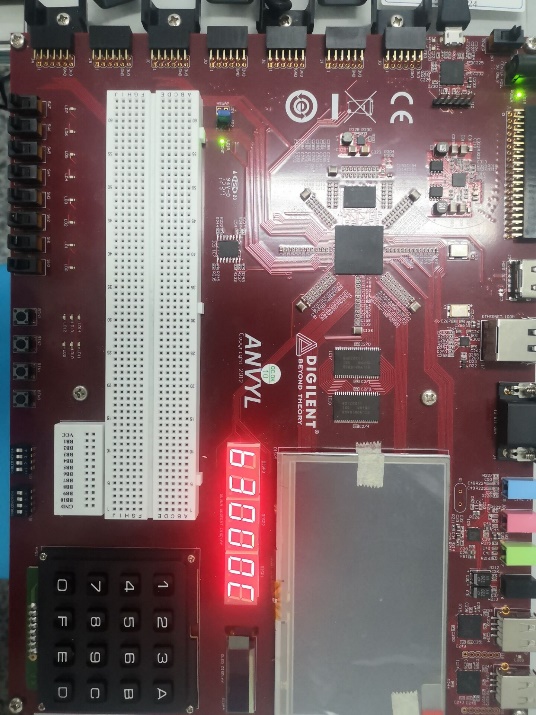


图8-43 PC=28，Inst=0xacb3000c

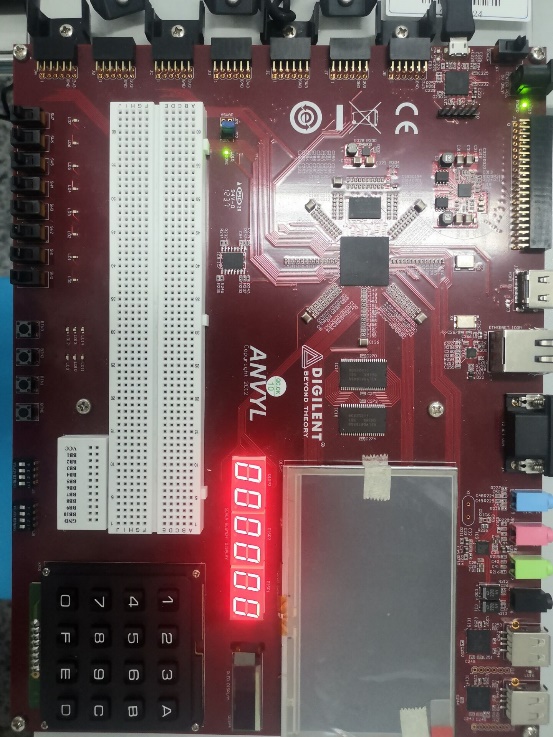


图8-44 PC=32，Inst=0x08000000

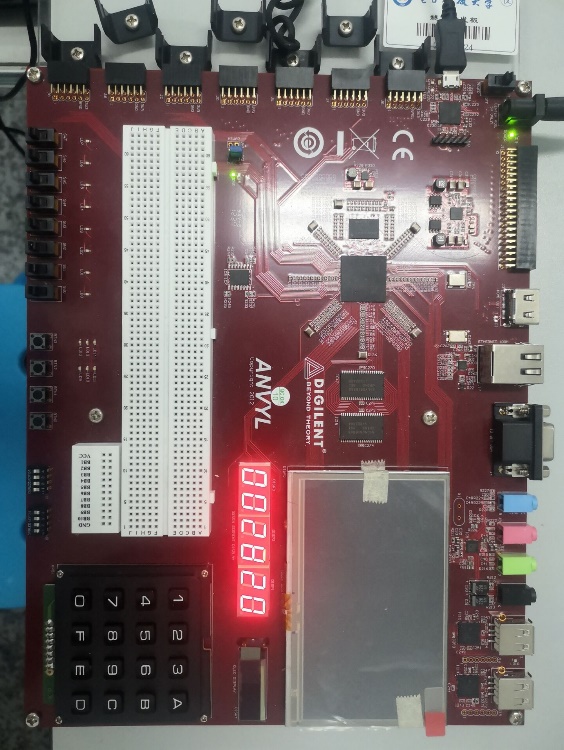


图8-45 PC=0，Inst=0x00002820

**结果分析：**

如表8-9所示，在第一个脉冲到达前，PC=0，Inst=0x00002820，置$a1为0；在第一个脉冲到达后，PC=4，Inst=0x8Cb10000，读Mem[0]到$s1；在第二个脉冲到达后，PC=8，Inst=0x8cb20004,读Mem[4]到$s2；在第三个脉冲到达后，PC=12，Inst=0x02329822，置$s3为$s1-$s2；在第四个脉冲到达后，PC=16，Inst=0x02328820，置$s1为$s1+$s2；在第五个脉冲到达后，PC=20，Inst=0x8cb20008,读Mem[8]到$s2；在第六个脉冲到达后，PC=24，Inst=0x12320002,根据$s1和$s2进行条件跳转（由于$s1的值为5，$s2的值为6，不发生跳转）；在第七个脉冲到达后，PC=28，Inst=0xacb3000c,写$s3到Mem[12]；在第八个脉冲到达后，PC=32，Inst=0x08000000,无条件跳转，构成循环；在第九个脉冲到达后，PC=0，Inst=0x00002820，置$a1为0，又回到了起点。

表8-9 板级测试结果分析

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 脉冲 | PC | Inst | Aluout | B\_data | $s1,$s2,$s3 | Mem[0,4,8,12] |
| 0 | 0 | 0x00002820 | 0 | 0 | 17,18,19 | 2,3,6,x |
| 1 | 4 | 0x8Cb10000 | 0 | 17 | 2,18,19 | 2,3,6,x |
| 2 | 8 | 0x8cb20004 | 4 | 18 | 2,3,19 | 2,3,6,x |
| 3 | 12 | 0x02329822 | -1 | 3 | 2,3,-1 | 2,3,6,x |
| 4 | 16 | 0x02328820 | 5 | 3 | 5,3,-1 | 2,3,6,x |
| 5 | 20 | 0x8cb20008 | 8 | 3 | 5,6,-1 | 2,3,6,x |
| 6 | 24 | 0x12320002 | -1 | 6 | 5,6,-1 | 2,3,6,x |
| 7 | 28 | 0xacb3000c | 12 | -1 | 5,6,-1 | 2,3,6,-1 |
| 8 | 32 | 0x08000000 | 0 | 0 | 5,6,-1 | 2,3,6,-1 |
| 9 | 0 | 0x00002820 | 0 | 0 | 5,6,-1 | 2,3,6,-1 |

据上述分析可知，在一个时钟周期内所设计的CPU能够完成一条指令的执行，指令执行结果与预期的结果是一致的。

（注：由于报告篇幅问题，仅给出Inst的输出测试结果和分析，PC，ALUout，B\_data的测试结果也均正确）

1. **实验结论：**

通过设计，实现，仿真测试和板级测试，成功实现了单周期CPU并完成了正确性验证。对于单周期CPU，在一个时钟周期完成指令的所有执行步骤，可以使CPU的设计简化，但由于没有考虑不同部件完成操作所需时间上的差异，故CPU各部件的利用率不高。

1. **总结及心得体会：**

**总结：**

该实验中，我首先实现并测试了二选一多路选择器、移位器、符号扩展器和加法器等基本功能模块。

在基本模块的基础上，我设计、实现并测试了算数逻辑单元，寄存器堆，30位取指部件，控制器（主控制单元和ALU局部控制单元）等相对复杂的功能模块。

然后按照数据通路的设计，我对上述功能模块进行了连接和封装，构成了较完整的主板模块。根据冯诺依曼存储程序和数据的思想，我又为主板添加了指令存储器（ROM）和数据存储器（RAM），构成了完整的主板模块，并进行了仿真测试。

进一步，我为主板添加了按键去抖电路和七段数码管驱动电路构成了单周期计算机，该计算机可以将外界按键去抖后作为时钟脉冲输入CPU，并且将内部信息显示到七段数码管上。最后，通过添加约束文件和二进制代码的烧录，我成功在开发板上验证了本次实验的成功和正确性。

附加地，在老师的要求和激励下，我完成了30位PC和J型指令的拓展题。

**心得体会：**

本次实验具有循序渐进的特点。首先通过正确地实现基本功能模块，我不仅复习了数字逻辑课中Verilog HDL的相关知识，也对CPU的内部设计和实现有了初步的认识。

本实验也有分而治之（模块）的特点。基于基本功能模块，我又分别实现了算数逻辑单元，寄存器堆，取指令部件，控制单元各模块，其中控制单元又可分解为主控制单元和ALU局部控制单元等更小的模块，大大降低了模块内的复杂度，提高了系统设计的可靠性。

本实验也有单元测试和小步快走的特点。在封装CPU和主板之前，我对各功能部件进行了仿真测试，以保持各模块功能的正确性；在封装CPU和主板之后，我也对CPU和主板进行了仿真测试，以此把BUG消除在萌芽阶段，大大减少了后期DEBUG的时间开销。

本实验也有理论结合实际的特点。实验中我不仅进行了仿真测试，还为主板添加了IO部件进行了板级测试。通过板级测试，让我通过实践进一步验证了设计和实现的正确性，很有成就感。同时按键去抖电路的存在让我意识到了理论与现实的鸿沟（物理世界的不完美性），七段数码管的使用让我对计算机驱动外设也有了更加深刻的认识。

最重要的是，对单周期CPU的实现和组装让我更加了解CPU的结构，为后续课程的学习打下了坚实的基础，收获颇丰。

1. **对本实验过程及方法、手段的改进建议：**
2. 可以实现更多类型的指令，比如移位指令，需要零扩展的I型指令，这样可以对CPU的数据通路实现有更加全面的认识。
3. 可以使用更多的指令进行测试，这样可以加深对各类指令工作流程的认识。
4. 特别地，可以考虑实现子过程调用与返回功能（作为拓展题）。函数是高级编程语言的基本概念之一，通过jal和jr指令，及保护寄存器等功能的实现，不仅可以提高该实验的趣味性和挑战性，同时还能让同学们对函数调用的本质有更加深刻的认识。

**报告评分：**

**指导教师签字：**