**电子科技大学信息与软件工程学院**

**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： 学 号： 指导教师：**

**实验地点： 实验时间：**

1. **实验室名称：**
2. **实验项目名称：**

单周期CPU的设计与实现。

1. **实验学时：**

实验学时是8学时。

1. **实验原理：**

本实验的单周期CPU的工作原理如下图1如示。

图1 单周期CPU总体电路

Clock

4

0 1 2 3

rs

op

rt

rd

func

Wreg

aluc

pcsource

S

sa

Inst

Mem

A

Do

+

ALU

A

B

A\_data

A\_addr

B\_addr

Write

W\_addr

B\_data

Data

f

S

PC

S

e

S

aluimm

shift

Result

Data

Mem

A

Do

We

S

wmem

m2reg

+

<<

Di

imm

addr

<<

Zero

regrt

sext

jal

Control

Unit

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20条，分为4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**

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。

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。

1. **实验内容**
2. 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20条。
3. CPU各功能模块的设计与实现；
4. CPU的封装；
5. 对所做设计的仿真测试和板级验证。
6. **实验器材（设备、元器件）：**
7. 安装了Xilinx ISE Design Suite 13.4的PC机一台
8. FPGA开发板：Anvyl Spartan6/XC6SLX45
9. 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1所示。

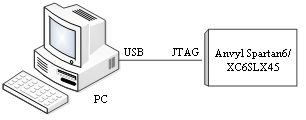


图1 PC计算机与FPGA开发板的连接

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CPU支持的指令共有20条MIPS，包含了R型、I型和J型指令，如表1所示。

表 1 本实验所涉及的20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 |
| Sll | 000000 | 00000 | rt | rd | sa | 000000 | 左移 |
| Srl | 000000 | 00000 | rt | rd | sa | 000010 | 逻辑右移 |
| Sra | 000000 | 00000 | rt | rd | sa | 000011 | 算术右移 |
| Jr | 000000 | rs | rt | rd | 000000 | 001000 | 寄存器跳 |
| I型指令 | | | | | | | |
| Addi | 001000 | rs | rt | immediate | | | 立即数加 |
| Andi | 001100 | rs | rt | immediate | | | 立即数与 |
| Ori | 001101 | rs | rt | immediate | | | 立即数或 |
| Xori | 001110 | rs | rt | immediate | | | 立即数异或 |
| 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 |
| Bne | 000101 | rs | rt | offset | | | 不等转移 |
| Lui | 001111 | 00000 | rt | immediate | | | 设置高位 |
| J型指令 | | | | | | | |
| J | 000010 | address | | | | | 跳转 |
| Jal | 000011 | address | | | | | 调用 |

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. 多路选择器的设计与实现

* 32位2选1多路选择器的设计与实现

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

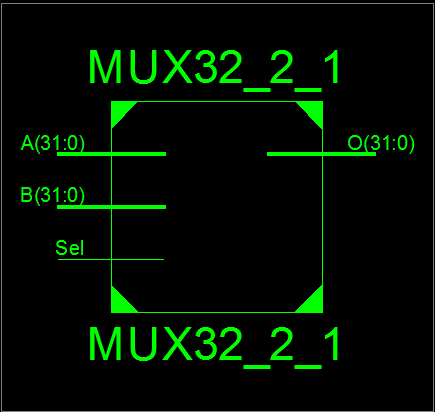
在ISE集成开发环境中，对模块MUX32\_2\_1进行综合（Synthesize），综合结果如图2所示。

图2 模块MUX32\_2\_1的综合（Synthesize）结果

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

module MUX32\_2\_1\_tb;

reg [31:0] A;

reg [31:0] B;

reg Sel;

wire [31:0] O;

// Instantiate the Unit Under Test (UUT)

MUX32\_2\_1 uut (

.A(A),

.B(B),

.Sel(Sel),

.O(O)

);

initial begin

// Initialize Inputs

A = 0;

B = 0;

Sel = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

A <= 32'haaaa0000;

B <= 32'hbbbb1111;

Sel <= 1'b0;

#100;

Sel <= 1'b1;

end

endmodule

然后进行仿真，仿真结果如图3所示。

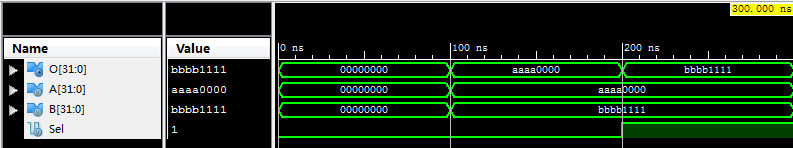


图3 模块MUX32\_2\_1的仿真结果

* 32位4选1多路选择器的设计与实现

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\_4\_1，然后输入其实现代码：

module MUX32\_4\_1(

input [31:0] A,

input [31:0] B,

input [31:0] C,

input [31:0] D,

input [1:0] Sel,

output[31:0] O

);

assign O = (Sel == 2'b00)? A : (Sel == 2'b01)? B :

(Sel == 2'b10)? C : D;

endmodule

在ISE集成开发环境中，对模块MUX32\_4\_1进行综合（Synthesize），综合结果如图4所示。

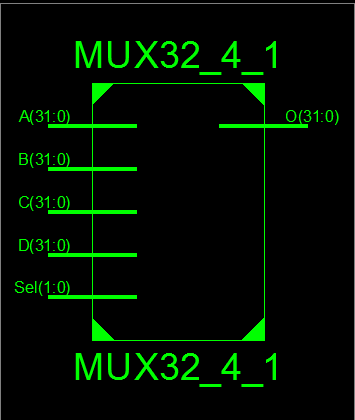


图4 模块MUX32\_4\_1的综合（Synthesize）结果

* 5位2选1多路选择器的设计与实现

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

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5所示。

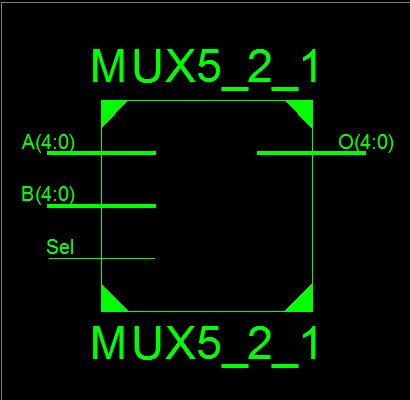


图5 模块MUX5\_2\_1的综合（Synthesize）结果

1. 32位D触发器的设计与实现

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：dff32，然后输入其实现代码：

module dff32(

input [31:0] D,

input Clock,

input Reset,

output reg [31:0] Q

);

always @(posedge Clock or negedge Reset) begin

if(Reset == 0) Q <= 0;

else Q <= D;

end

endmodule

在ISE集成开发环境中，对模块dff32进行综合（Synthesize），综合结果如图6所示。

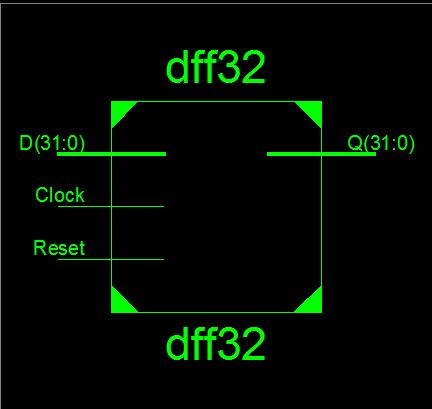


图6 模块dff32的综合（Synthesize）结果

1. 移位器的设计与实现

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：Shifter，然后输入其实现代码：

module Shifter(

input [31:0] D,

input [4:0] Sa,

input Right,

input Arith,

output reg [31:0] O

);

always @(\*) begin

if (!Right) O = D << Sa ;

else if (!Arith) O = D >> Sa;

else O = $signed(D) >>> Sa;

end

endmodule

在ISE集成开发环境中，对模块Shifter进行综合（Synthesize），综合结果如图7所示。

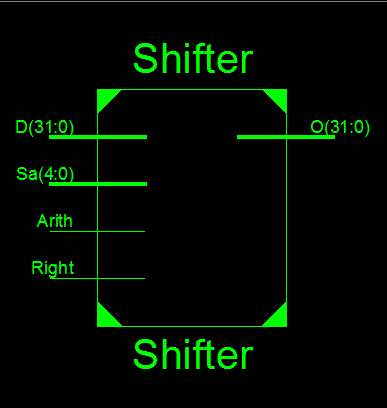


图7 模块Shifter的综合（Synthesize）结果

1. 32位加/减法器的设计与实现

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：addsub32，然后输入其实现代码：

module addsub32(

input [31:0] A,

input [31:0] B,

input sub,

output[31:0] Result

);

cla32 as32(A , B^{32{sub}} , sub , Result);

endmodule

module cla32(

input [31:0] a,

input [31:0] b,

input c,

output[31:0] s

);

assign s = a + b + c;

endmodule

在ISE集成开发环境中，对模块addsub32进行综合（Synthesize），综合结果如图8所示。

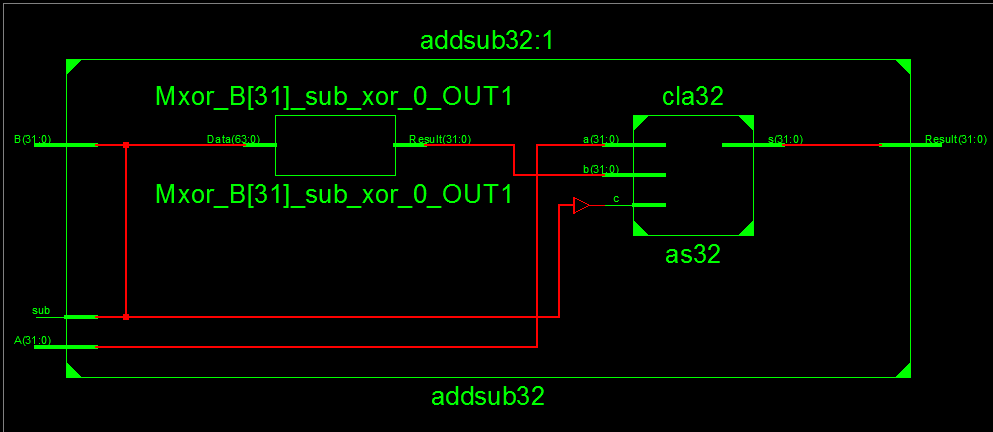


图8 模块addsub32的综合（Synthesize）结果

1. 运算器（ALU）设计与实现

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,

input [31:0] B,

input [3:0] ALU\_Operation,

output [31:0] Result,

output Zero

);

wire [31:0] d\_and = A & B;

wire [31:0] d\_or = A | B;

wire [31:0] d\_xor = A ^ B;

wire [31:0] d\_lui = {B[15:0],16'h0};

wire [31:0] d\_and\_or = ALU\_Operation[2]? d\_or:d\_and;

wire [31:0] d\_xor\_lui = ALU\_Operation[2]? d\_lui:d\_xor;

wire [31:0] d\_as , d\_sh;

addsub32 as32 (A , B , ALU\_Operation[2] , d\_as);

Shifter shift\_1 ( B , A[4:0] , ALU\_Operation[2] ,

ALU\_Operation[3] , d\_sh);

MUX32\_4\_1 sel ( d\_as , d\_and\_or , d\_xor\_lui ,

d\_sh , ALU\_Operation[1:0] , Result);

assign Zero = ~|Result;

endmodule

在ISE集成开发环境中，对模块ALU进行综合（Synthesize），综合结果如图9所示。

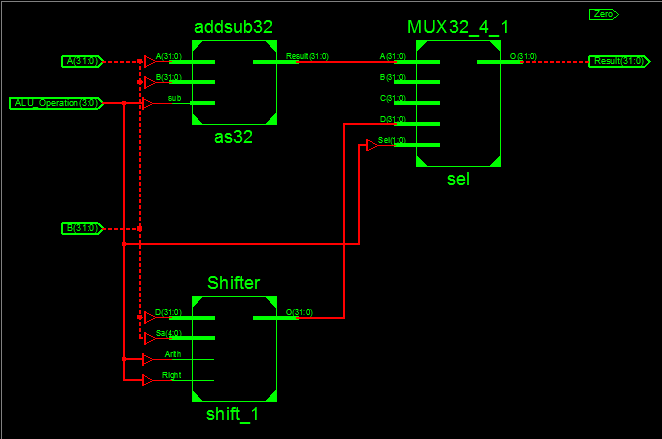


图9 模块ALU的综合（Synthesize）结果

1. 寄储器堆（Register File）设计与实现

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 Clock,

input Reset,

input [ 4:0] A\_addr,

input [ 4:0] B\_addr,

input [ 4:0] W\_addr,

input [31:0] Data,

input Write,

output [31:0] A\_data,

output [31:0] B\_data

);

reg [31:0] Register[1:31];

assign A\_data = (A\_addr == 0)? 0 : Register[A\_addr];

assign B\_data = (B\_addr == 0)? 0 : Register[B\_addr];

integer i;

always @ ( posedge Clock or negedge Reset) begin

if (Reset == 0) begin

for (i=1 ; i <=31 ; i = i+1) Register[i] <= 0;

end else if (( Write ) && ( W\_addr != 0))

Register[W\_addr] <= Data;

end

endmodule

在ISE集成开发环境中，对模块RegFile进行综合（Synthesize），综合结果如图10所示。

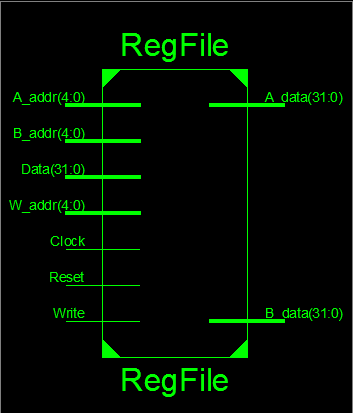


图10 模块RegFile的综合（Synthesize）结果

1. 控制器（Control Unit）设计与实现

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,

input [5:0] func,

input z,

output wmem,

output wreg,

output regrt,

output m2reg,

output [3:0] aluc,

output shift,

output aluimm,

output [1:0] pcsource,

output jal,

output sext

);

wire i\_add = (op == 6'b000000 & func == 6'b100000)?1:0;

wire i\_sub = (op == 6'b000000 & func == 6'b100010)?1:0;

wire i\_and = (op == 6'b000000 & func == 6'b100100)?1:0;

wire i\_or = (op == 6'b000000 & func == 6'b100101)?1:0;

wire i\_xor = (op == 6'b000000 & func == 6'b100110)?1:0;

wire i\_sll = (op == 6'b000000 & func == 6'b000000)?1:0;

wire i\_srl = (op == 6'b000000 & func == 6'b000010)?1:0;

wire i\_sra = (op == 6'b000000 & func == 6'b000011)?1:0;

wire i\_jr = (op == 6'b000000 & func == 6'b001000)?1:0;

wire i\_addi = (op == 6'b001000)?1:0;

wire i\_andi = (op == 6'b001100)?1:0;

wire i\_ori = (op == 6'b001101)?1:0;

wire i\_xori = (op == 6'b001110)?1:0;

wire i\_lw = (op == 6'b100011)?1:0;

wire i\_sw = (op == 6'b101011)?1:0;

wire i\_beq = (op == 6'b000100)?1:0;

wire i\_bne = (op == 6'b000101)?1:0;

wire i\_lui = (op == 6'b001111)?1:0;

wire i\_j = (op == 6'b000010)?1:0;

wire i\_jal = (op == 6'b000011)?1:0;

assign wreg = i\_add | i\_sub | i\_and | i\_or |

i\_xor | i\_sll | i\_srl |i\_sra |

i\_addi | i\_andi | i\_ori | i\_or |

i\_xori | i\_lw | i\_lui |i\_jal;

assign regrt = i\_addi | i\_andi | i\_ori | i\_xori |

i\_lw | i\_lui;

assign jal = i\_jal;

assign m2reg = i\_lw;

assign shift = i\_sll | i\_srl |i\_sra;

assign aluimm = i\_addi | i\_andi | i\_ori | i\_xori |

i\_lw | i\_lui |i\_sw;

assign sext = i\_addi | i\_lw | i\_sw | i\_beq | i\_bne;

assign aluc[3] = i\_sra;

assign aluc[2] = i\_sub | i\_or | i\_srl | i\_sra |

i\_ori | i\_lui;

assign aluc[1] = i\_xor | i\_sll | i\_srl | i\_sra |

i\_xori | i\_beq | i\_bne | i\_lui;

assign aluc[0] = i\_and | i\_or | i\_sll | i\_srl |i\_sra |

i\_andi | i\_ori;

assign wmem = i\_sw;

assign pcsource[1] = i\_jr | i\_j | i\_jal;

assign pcsource[0] = i\_beq & z | i\_bne&~z | i\_j | i\_jal;

endmodule

在ISE集成开发环境中，对模块Control\_Unit进行综合（Synthesize），综合结果如图11所示。

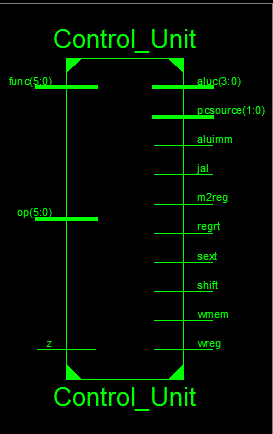


图11 模块Control\_Unit的综合（Synthesize）结果

1. CPU的封装

CPU封装的基本原理就是根据单单周期CPU的总体电路将各功能部件连接起来按照指令的数据路径连接起来。

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 Clock,

input Reset,

input [31:0] inst,

input [31:0] mem,

output[31:0] pc,

output wmem,

output[31:0] Result,

output[31:0] Data

);

wire [31:0] p4 , adr , npc , res , ra , alu\_mem , alua , alub;

wire [4:0] reg\_dest , wn;

wire [3:0] aluc;

wire [1:0] pcsource;

wire zero , wreg,regrt, m2reg,shift,aluimm,jal,sext;

wire [31:0] sa = {27'b0 , inst[10:6]};

wire [31:0] offset = {imm[13:0] , inst[15:0] , 2'b00};

dff32 ip (npc , Clock , Reset , pc);

cla32 pcplus4 ( pc , 32'h4 , 1'b0 , p4);

cla32 br\_adr ( p4 , offset , 1'b0 ,adr);

wire [31:0] jpc = {p4[31:28] , inst[25:0],2'b00};

Control\_Unit CU ( inst[31:26] , inst[5:0] , zero,

wmem,wreg,regrt,m2reg,aluc,

shift,aluimm,pcsource,jal,sext );

wire e =sext & inst[15];

wire [15:0] imm = {16{e}};

wire [31:0] immdiate = {imm , inst[15:0]}; //立即数

RegFile rf ( Clock, Reset, inst[25:21], inst[20:16],

wn, res, wreg, ra, Data );

MUX32\_2\_1 alu\_a ( ra , sa , shift , alua);

MUX32\_2\_1 alu\_b ( Data , immdiate , aluimm , alub);

ALU alu( alua , alub , aluc , Result , zero);

MUX5\_2\_1 reg\_wn ( inst[15:11] , inst[20:16] , regrt ,

reg\_dest);

assign wn = reg\_dest | {5{jal}};

MUX32\_2\_1 res\_mem ( Result , mem , m2reg , alu\_mem);

MUX32\_2\_1 link ( alu\_mem , p4 , jal , res);

MUX32\_4\_1 nextpc( p4 , adr , ra , jpc , pcsource , npc);

endmodule

在ISE集成开发环境中，对模块Data\_flow进行综合（Synthesize），综合结果如图12所示。

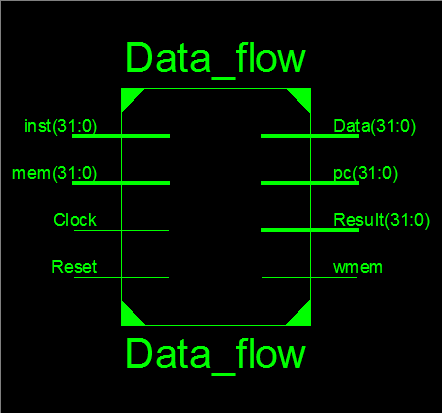


图12 Data\_flow模块的综合（Synthesize）结果

1. 测试
2. 指令存储器

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\_mem，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\_mem，单击Next进入下一步，点击Finish完成创建。实现Inst\_mem的Verilog程序参考代码：

module Inst\_mem(

input [31:0] address,

output [31:0] inst

);

wire [31:0] ram [0:31];

assign ram[5'h00] = 32'h3c011234; //lui R1 , 0x1234

assign ram[5'h01] = 32'h34215678; //ori R1 , R1 ,0x5678

assign ram[5'h02] = 32'h3C02AAAA; //lui R2 , 0xaaaa

assign ram[5'h03] = 32'h3442BBBB; //ori R2 , R2 ,0xbbbb

assign ram[5'h04] = 32'h00221820; //add R3 , R1 , R2

assign ram[5'h05] = 32'h00221822; //sub R3 , R1 , R2

assign ram[5'h06] = 32'h00221824; //and R3 , R1 , R2

assign ram[5'h07] = 32'h00221825; //or R3 , R1 , R2

assign ram[5'h08] = 32'h00221826; //xor R3 , R1 , R2

assign ram[5'h09] = 32'h00021900; //sll R3 , R2 , 4

assign ram[5'h0a] = 32'h00021902; //srl R3 , R2 , 4

assign ram[5'h0b] = 32'h00021903; //srl R3 , R2 , 4

assign ram[5'h0c] = 32'h20231234; //addi R3 , R1,0x1234

assign ram[5'h0d] = 32'h302300EF; //andi R3 , R1,0xef

assign ram[5'h0e] = 32'h342300EF; //ori R3 , R1,0xef

assign ram[5'h0f] = 32'h382300EF; //xori R3 , R1,0xef

assign ram[5'h10] = 32'h00631826; //xor R3 , R3 , R3

assign ram[5'h11] = 32'hAC610001; //sw R1 , 1(R3)

assign ram[5'h12] = 32'h90650001; //lw R5 , 1(R3)

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

endmodule

1. 指令存储器

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\_mem，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 \_mem，单击Next进入下一步，点击Finish完成创建。实现Data \_mem的Verilog程序参考代码：

module Data\_mem(

input Clock,

output[31:0] dataout,

input [31:0] datain,

input [31:0] addr,

input we,

input inclk,

input outclk

);

reg [31:0] ram [0:31];

assign dataout = ram[addr[6:2]];

always @ (posedge Clock) begin

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

end

integer i;

initial begin

for ( i = 0 ; i <= 31 ; i = i + 1) ram [i] = i \* i;

end

endmodule

1. 仿真测试

将CPU、指令存储器和数据存储器连接起来。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Main\_board，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 Clock,

input Reset,

input mem\_clk,

output[31:0] inst,

output[31:0] pc,

output[31:0] aluout,

output[31:0] memout

);

wire [31:0] data;

wire wmem;

Data\_flow CPU (Clock , Reset , inst , memout , pc , wmem , aluout ,data);

Inst\_mem imem(pc , inst);

Data\_mem dmem(Clock , memout , data , aluout , wmem , Clock , Clock);

Endmodule

其连接如图13所示。

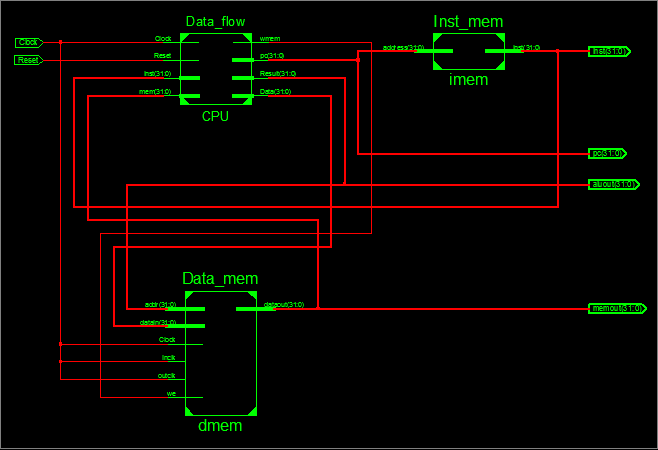


图13 CPU、指令存储器和数据存储器连接图

向工程中添加测试模块。综合通过后，在工程管理区将View设置为Simulation，在任意位置单击鼠标右键，并在弹出的菜单中选择New Source，在类型中选择Verilog Test Fixture，输入测试文件名：Mainboard\_tb，单击下一步。这时所有工程中的模块名都会显示出来，选择要进行测试的模块：Mainboard。点击Next ,再单击Finish按钮，ISE会在源代码编辑区自动生成测试模块的代码。我们看到，ISE已经自动生成了基本的信号并对被测模块做了例化。

在Mainboard\_tb添加如下测试代码：

initial begin

// Initialize Inputs

Clock = 0;

Reset = 0;

mem\_clk = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

Reset <= 1;

end

always begin

#50;

Clock = ~Clock;

End

完成测试文件编辑后，确认工程管理区中View选项设置为Simulation，并且选中的测试模块是Mainboard\_tb，这时在过程管理区会显示与仿真有关的进程。右键单击其中的Simulate Behavioral Model项，选择弹出菜单中的Process Properties项，会弹出属性设置对话框，将其中Simulation Run Time设置为1000ns。

在过程管理区双击Simulate Behavioral Model，ISE将启动ISE Simulator，可以得到仿真结果，如图14所示。

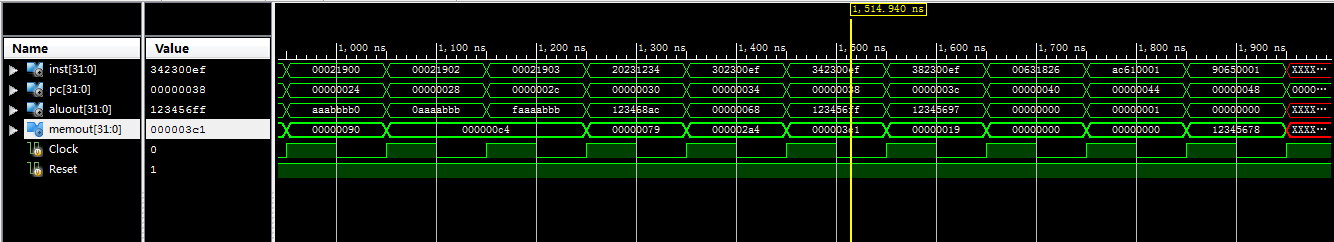
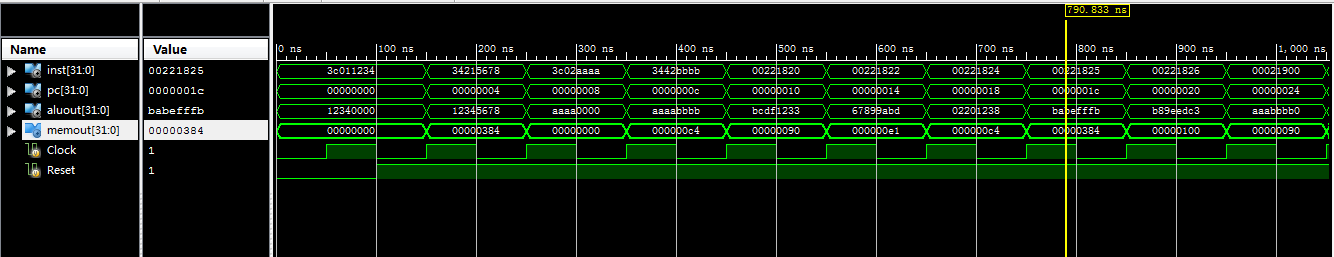


图14 CPU仿真测试结果

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；
* 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 – 显示数据存储器输出
* 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

对模块Mainboard作如下修改：

module Mainboard(

input Clock,

input Reset,

output[31:0] inst,

output[31:0] pc,

output[31:0] aluout,

output[31:0] memout,

input BTN\_IN, //单步脉冲输入

input [5:0] SW, //开关

output [7:0] seg, //数码管7段

output [5:0] AN\_SEL,

output [7:0] LED

);

wire [31:0] data;

wire wmem;

Data\_flow CPU (Clock , Reset , inst , memout , pc , wmem , aluout ,data);

Inst\_mem imem(pc , inst);

Data\_mem dmem(Clock , memout , data , aluout , wmem );

//单步执行时钟控制

reg [31:0] clockdiv = 0;

wire BTN\_Out;

BTN\_Anti\_Jitter anti\_jitter( clockdiv[12] , BTN\_IN , BTN\_Out ); //按钮去抖动

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

assign LED[0] = BTN\_Out | Reset;

assign LED[1] = SW[1];

assign LED[2] = SW[2];

//控制数码管显示

wire [31:0] display\_content;

wire [23:0] disp\_num;

//显示内容选择

assign display\_content = (SW[1:0] == 2'b00)? inst:

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

(SW[1:0] == 2'b10)? aluout : memout;

//高低24位选择

assign disp\_num = (SW[2] == 1)? display\_content[31:8]:

display\_content[23:0];

//数码管输出显示

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

endmodule

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，弹出如图4-6所示对话框，在类型中选择Implementation Constraints File，输入文件名：Mainboard\_ucf，选择关联模块：Mainboard。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。Mainboard.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)

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

#switches

NET SW<0> LOC= V5;

NET SW<1> LOC= U4;

NET SW<2> LOC= V3;

#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

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Mainboard.bit文件。将生成的Mainboard.bit下载开发板上验证

1. **实验数据及结果分析：**

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。

1. **实验结论：**

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：**
2. **对本实验过程及方法、手段的改进建议：**

**报告评分：**

**指导教师签字：**