计算机组成原理实验4

PB20000156
徐亦昶

数据通路设计

ppt中数据通路少了很多指令的支持，先对其进行扩充，扩充后的数据通路支持24条指令（要求的10条、大部分寄存器运算指令、大部分立即数运算指令、分支指令）。
数据通路如下：
 本模块不包括PDU接口，但稍作修改接PDU很容易。各模块控制原理：

Control

Opcode	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	Jalr	Aquipc	Jal	RegWrite
0010011	0	0	0	11	0	1	0	0	0	1
0110011	0	0	0	10	0	0	0	0	0	1
0010111	0	0	0	11	0	0	0	1	0	1
1101111	0	0	0	11	0	0	0	0	1	1
1100111	0	1	0	11	0	1	1	0	0	1
1100011	1	0	0	01	0	0	0	0	0	0
0000011	0	1	1	00	0	1	0	0	0	1
0100011	0	0	0	00	1	1	0	0	0	0

ALU control

ALUOp	instruction[30,14-12]	ALU_sel
00	xxxx	001(+)
01	xxxx	000(-)
10	0000	001(+)
10	1000	000(-)
10	0001	110(<<)
10	0100	100(^)
10	0101	101(>>)
10	1101	111(>>>)
10	0110	011(or)
10	0111	010(and)
11	x000	001(+)
11	x100	100(^)
11	x110	011(or)

ALUOp	instruction[30,14-12]	ALU_sel
11	x111	010(and)

Imm Gen

instruction[6:0]	Imm
0010011	{{20{instruction[31]}},instruction[31:20]}
1100011	{{20{instruction[31]}},instruction[7],instruction[30:25],instruction[11:8]}
0000011	{{20{instruction[31]}},instruction[31:20]}
0100011	{{20{instruction[31]}},instruction[31:25],instruction[12:8]}
others	32'b0

PCSrc driver

Branch	Zero	PCSrc
0	xxxxxx	0
1	000001(beq)	1
1	000xx0	0
1	001xx0(bne)	1
1	001001	0
1	100x10(blt)	1
1	100x0x	0
1	101x0x(bge)	1
1	101x10	0
1	1101x0(bltu)	1
1	1100xx	0
1	1110xx(bgeu)	1
1	1111x0	0

PC Ctrl2

当指令为aquipc或jal时对PC值进行运算。

instruction[6:3]	Out
0010(aquipc)	PC + sext(instruction[31:12]<<12)<<1
1101(jal)	pc + sext(offset)
others	32'b0

代码编写

CPU包含两个单端口RAM：dist_mem_gen_0/1，其中0是数据存储器，1是指令存储器。它们的Depth和Data Width均设为12288和32，初始化coe文件分别为xxx_data.coe和xxx_text.coe，分别表示程序数据段和代码段。

Add

执行加法操作。模块化是为了方便调试。

```
//Add.v
`timescale 1ns / 1ps
module Add #(
parameter WIDTH=32
)(
input[WIDTH-1:0] a,b,
output[WIDTH-1:0] y);
assign y=a+b;
endmodule
```

仿真文件：

```
//test_Add.v
`timescale 1ns / 1ps
module test_Add();
reg[31:0] a,b;
wire[31:0] y;
Add Add(a,b,y);
initial
begin
    a=32'hac;
    b=32'hb1;
    #1 $finish;
end
endmodule
```

MUX

32位数据选择器

```
//MUX.v
`timescale 1ns / 1ps
module MUX(
input[31:0] s0,s1,
input sel,
output[31:0] d);
assign d=sel?s1:s0;
endmodule
```

MUX5

5位数据选择器

```
//MUX5.v
`timescale 1ns / 1ps
module MUX5(
input[4:0] s0,s1,
input sel,
output[4:0] d);
assign d=sel?s1:s0;
endmodule
```

PC_driver

用于产生新的PC

```
//PC_driver.v
`timescale 1ns / 1ps
module PC_driver(
input[31:0] NPC,
input clk,rstn,
output reg[31:0] PC);
always@(posedge clk or negedge rstn)
begin
    if(rstn==0)
        PC<=32'h3000;
    else
        PC<=NPC;
end
endmodule
```

ALU

同实验1的ALU。

```
//ALU.v
`timescale 1ns / 1ps
module ALU #(
parameter WIDTH=32
)(
input [WIDTH-1:0] a,b,
input [2:0] s,
output reg [WIDTH-1:0] y,
output reg [2:0] f
);
always@(*)
begin
    if(s==3'b000)
    begin
        f[0]=(a==b)?1:0;
        f[2]=(a<b)?1:0;
        case({a[WIDTH-1],b[WIDTH-1]})
            2'b00:
                f[1]=(a<b)?1:0;
            2'b01:
                f[1]=0;
            2'b10:
                f[1]=1;
            2'b11:
                f[1]=(a[WIDTH-2:0]<b[WIDTH-2:0])?1:0;
            default:
                f[1]=0;
        endcase
    end
    else
        f=3'b000;
end
always@(*)
begin
    case(s)
        3'b000:y=a-b;
        3'b001:y=a+b;
        3'b010:y=a&b;
        3'b011:y=a|b;
        3'b100:y=a^b;
        3'b101:y=a>>b;
        3'b110:y=a<<b;
        3'b111:y=a>>>b;
        default:y=a;
    endcase
end
endmodule
```

instruction_memory

指令存储器

```
//instruction_memory.v
`timescale 1ns / 1ps
module instruction_memory(
input clk,
input[31:0] PC,
output[31:0] instruction);
wire [31:0] shifted=(PC-32'h3000)>>2;
dist_mem_gen_1 IM (
      .a(shifted[13:0]),      // input wire [13 : 0] a
      .d(0),      // input wire [31 : 0] d
      .clk(clk),  // input wire clk
      .we(0),    // input wire we
      .spo(instruction)  // output wire [31 : 0] spo
    );
endmodule
```

registers

寄存器堆

```
//registers.v
`timescale 1ns / 1ps
module registers #(
    parameter AW=5,
    parameter DW=32
)(
    input clk,
    input [AW-1:0] ra0,ra1,
    output [DW-1:0] rd0,rd1,
    input [AW-1:0] wa,
    input [DW-1:0] wd,
    input we
);
reg [DW-1:0] rf [0:(1<<AW)-1];
assign rd0=rf[ra0];
assign rd1=rf[ra1];
initial
rf[0]=0;
always@(posedge clk)
if(we&&wa!=0)
    rf[wa]=wd;
endmodule
```

PC_Ctrl2

针对部分指令对PC值进行运算。

```
//PC_Ctrl2.v
`timescale 1ns / 1ps
module PC_Ctrl2(
input[31:0] instruction,PC,
output reg[31:0] Out);
always@(*)
begin
    case(instruction[6:3])
        4'b0010:Out=PC+{instruction[31:12],12'b0};
        4'b1101:Out=PC+
{{12{instruction[31]}},instruction[19:12],instruction[20],instruction[30:21],1'b0};
        default:Out=32'b0;
    endcase
end
endmodule
```

Imm_Gen

生成立即数。

```
//Imm_Gen.v
`timescale 1ns / 1ps
module Imm_Gen(
input[31:0] instruction,
output reg[31:0] Imm);
always@(*)
begin
    casex(instruction[6:0])
        7'b0010011:Imm={{20{instruction[31]}},instruction[31:20]};
        7'b1100111:Imm={{20{instruction[31]}},instruction[31:20]};
        7'b1100011:Imm=
{{20{instruction[31]}},instruction[7],instruction[30:25],instruction[11:8]};
        7'b0000011:Imm={{20{instruction[31]}},instruction[31:20]}; //lw
        7'b0000011:Imm={{20{instruction[31]}},instruction[31:25],instruction[11:8]}; //sw
        default:Imm=32'b0;
    endcase
end
endmodule
```

ALU_control

根据指令和ALUOp判断ALU要做的运算类型。

```
//ALU_control
`timescale 1ns / 1ps
module ALU_control(
input[1:0] ALUOp,
input[3:0] instruction,
output reg[2:0] ALU_sel);
always@(*)
begin
    case(ALUOp)
        2'b00:ALU_sel=3'b001;
        2'b01:ALU_sel=3'b000;
        2'b10:
        begin
            case(instruction)
                4'b0000:ALU_sel=3'b001;
                4'b1000:ALU_sel=3'b000;
                4'b0001:ALU_sel=3'b110;
                4'b0100:ALU_sel=3'b100;
                4'b0101:ALU_sel=3'b101;
                4'b1101:ALU_sel=3'b111;
                4'b0110:ALU_sel=3'b011;
                4'b0111:ALU_sel=3'b010;
                default:ALU_sel=3'b000;
            endcase
        end
        2'b11:
        begin
            casex(instruction)
                4'bx000:ALU_sel=3'b001;
                4'bx100:ALU_sel=3'b100;
                4'bx110:ALU_sel=3'b011;
                4'bx111:ALU_sel=3'b010;
                default:ALU_sel=3'b000;
            endcase
        end
        default:ALU_sel=3'b000;
    endcase
end
endmodule
```

仿真文件：

```
//test_ALU_control.v
`timescale 1ns / 1ps
module test_ALU_control();
```

```
reg[1:0] ALUOp;
reg[3:0] instruction;
wire[2:0] ALU_sel;
initial
begin
    ALUOp=2'b00;
    instruction=4'b0000;
end
initial
begin
    #1 ALUOp=2'b01;
    #1 ALUOp=2'b10;
    #1 instruction=4'b1000;
    #1 instruction=4'b0001;
    #1 instruction=4'b0100;
    #1 instruction=4'b0101;
    #1 instruction=4'b1101;
    #1 instruction=4'b0110;
    #1 instruction=4'b0111;
    #1 ALUOp=2'b11;
    instruction=4'b0000;
    #1 instruction=4'b1000;
    #1 instruction=4'b1100;
    #1 instruction=4'b1110;
    #1 instruction=4'b1111;
    #1 $finish;
end
ALU_control ALU_control(ALUOp,instruction,ALU_sel);
endmodule
```

把每类指令和ALUOp都尝试一下，发现结果均正确。

data_memory

数据存储器。里面实现了MMIO。当检测到访问的是内存映射地址时，把数据的控制转交给外设。

```
//data_memory.v
`timescale 1ns / 1ps
module data_memory(
input MemWrite,MemRead,clk,
input [31:0]Address,WriteData,io_din,
output reg [31:0]ReadData,io_dout,
output reg[7:0] io_addr,
output reg io_we,io_rd);
wire[31:0] shifted=Address>>2;
wire[31:0] RealData;
reg RealWe;
dist_mem_gen_0 memory (
  .a(shifted[13:0]),      // input wire [15 : 0] a
  .d(WriteData),      // input wire [31 : 0] d
  .clk(clk),  // input wire clk
  .we(RealWe),    // input wire we
  .spo(RealData)  // output wire [31 : 0] spo
);
always@(*)
begin
    if(Address>=32'hff00)
```

```
        {ReadData,RealWe,io_dout,io_addr,io_we,io_rd}=
{io_din,1'b0,WriteData,Address[7:0],MemWrite,MemRead};
    else
        {ReadData,RealWe,io_dout,io_addr,io_we,io_rd}={RealData,MemWrite,32'b0,8'b0,2'b0};
end
endmodule
```

仿真:

```
//test_data_memory.v
`timescale 1ns / 1ps
module test_data_memory();
reg[31:0] a;
reg[31:0] d;
reg clk,we;
wire[31:0] spo;

data_memory data_memory(we,1'b1,clk,a,d,0,spo);

initial
begin
    clk=0;
    we=0;
    d=0;
    a=0;
    #2 we=1'b1;
    a=32'h4;
    d=32'haaaa;
    #2 a=0;
    #0.5 a=32'h4;
end

initial
forever #1
clk=~clk;
endmodule
```

PCSrc_driver

控制branch指令的跳转与否。

```
//PCSrc_driver.v
`timescale 1ns / 1ps
module PCSrc_driver(
input Branch,
input[5:0] Zero,
output reg PCSrc);
always@(*)
begin
    if(Branch==1'b0)
        PCSrc=1'b0;
    else
    begin
        casex(Zero)
            6'b000001:PCSrc=1'b1;
```

```
                6'b000xx0:PCSrc=1'b0;
                6'b001xx0:PCSrc=1'b1;
                6'b001001:PCSrc=1'b0;
                6'b100x10:PCSrc=1'b1;
                6'b100x0x:PCSrc=1'b0;
                6'b101x0x:PCSrc=1'b1;
                6'b101x10:PCSrc=1'b0;
                6'b1101x0:PCSrc=1'b1;
                6'b1100xx:PCSrc=1'b0;
                6'b1110xx:PCSrc=1'b1;
                6'b1111x0:PCSrc=1'b0;
                default:PCSrc=1'b0;
            endcase
        end
    end
    endmodule
```

仿真文件：

```
//test_PCSrc_driver.v
`timescale 1ns / 1ps
module test_PCSrc_driver();
reg Branch;
reg[5:0] Zero;
wire PCSrc;
initial
begin
    Branch=0;
    #1 Branch=1'b1;
    Zero=6'b000001;
    #1 Zero=6'b000000;
    #1 Zero=6'b001010;
    #1 Zero=6'b001001;
    #1 Zero=6'b100110;
    #1 Zero=6'b100001;
    #1 Zero=6'b100011;
    #1 Zero=6'b101101;
    #1 Zero=6'b101010;
    #1 Zero=6'b110100;
    #1 Zero=6'b110001;
    #1 Zero=6'b111010;
    #1 Zero=6'b111110;
    #1 $finish;
end
PCSrc_driver PCSrc_driver(Branch,Zero,PCSrc);
endmodule
```

所有输入组合都尝试了一遍，输出都正确。

chk_data_driver

控制调试输出数据。

```
//chk_data_driver
`timescale 1ns / 1ps
```

```
module chk_data_driver(
input[15:0] chk_addr,
input[31:0] NPC,PC,instruction,a,b,imm,y,mdr,
input[10:0] ctrl,
output reg[31:0] chk_data);
always@(*)
begin
    if(chk_addr[15:12]==4'h0)
    begin
        case(chk_addr[3:0])
            4'h0:chk_data=NPC;
            4'h1:chk_data=PC;
            4'h2:chk_data=instruction;
            4'h3:chk_data={21'b0,ctrl};
            4'h4:chk_data=a;
            4'h5:chk_data=b;
            4'h6:chk_data=imm;
            4'h7:chk_data=y;
            4'h8:chk_data=mdr;
            default:chk_data=32'b0;
        endcase
    end
    else if(chk_addr[15:12]==4'h1)
        chk_data=a;
    else
        chk_data=mdr;
end
endmodule
```

control

控制信号的发出模块。

```
//control.v
`timescale 1ns / 1ps
module control(
input[6:0] Opcode,
output reg Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite,
output reg[1:0] ALUOp);
always@(*)
begin
    case(Opcode)
        7'b0010011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b00011010001;
        7'b0110011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b00010000001;
        7'b0010111:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b00011000101;
        7'b1101111:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b00011000011;
        7'b1100111:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b01011011001;
        7'b1100011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b10001000000;
        7'b0000011:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b01100010001;
        7'b0100011:
```

```
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b00000110000;
        default:
{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite}=11'b0;
    endcase
end
endmodule
```

仿真文件：

```
//test_control.v
`timescale 1ns / 1ps
module test_control();
reg[6:0] Opcode;
wire Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite;
wire[1:0] ALUOp;
initial
Opcode=0;
initial
begin
    #1 Opcode=7'b0010011;
    #1 Opcode=7'b0110011;
    #1 Opcode=7'b0010111;
    #1 Opcode=7'b1101111;
    #1 Opcode=7'b1100111;
    #1 Opcode=7'b1100011;
    #1 Opcode=7'b0000011;
    #1 Opcode=7'b0100011;
    #1 $finish;
end
control
control(Opcode,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite,ALUOp);
endmodule
```

经检测，对于所有的指令输入，输出均和真值表一致。

CPU

把上面的模块串起来。

```
//CPU.v
`timescale 1ns / 1ps
module CPU(
input clk,rst,
output[7:0] io_addr,
output[31:0] io_dout,
output io_we,io_rd,
input[31:0] io_din,
output[31:0] chk_pc,
input[15:0] chk_addr,
output[31:0] chk_data);
wire[31:0]
PC,NPC,instruction,Out,PC_add4,Imm,PC_offseted,Jumped,Branched,ReadData1,ReadData2,RegWrite
Data,ReadMemory,ALUResult,ALUIn2,MUXd2Out,MUXd3Out,DebugMemData,DebugRegData,MemAddr;
wire
Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite,PCSrc,DebugMem,DebugReg;
```

```
wire[1:0] ALUOp;
wire[2:0] ALUZero,ALUControlSignal;
wire[4:0] RegAddr;
assign DebugMem=chk_addr[13]?1'b1:1'b0;
assign DebugReg=chk_addr[12]?1'b1:1'b0;
assign chk_pc=NPC;
PC_driver PC_driver(.NPC(NPC),.clk(clk),.rstn(~rst),.PC(PC));
instruction_memory instruction_memory(.clk(clk),.PC(PC),.instruction(instruction));
Add Add1(.a(PC),.b(32'h4),.y(PC_add4)); //Add PC by 4
Add Add2(.a(PC),.b(Imm<<1),.y(PC_offseted)); //Add PC by offset
ALU ALU(.a(ReadData1),.b(ALUIn2),.s(ALUControlSignal),.y(ALUResult),.f(ALUZero));
control
control(.Opcode(instruction[6:0]),.Branch(Branch),.MemRead(MemRead),.MemtoReg(MemtoReg),.MemWrite(MemWrite),.ALUSrc(ALUSrc),.Jalr(Jalr),.Aquipc(Aquipc),.Jal(Jal),.RegWrite(RegWrite),.ALUOp(ALUOp));
registers
registers(.clk(clk),.ra0(RegAddr),.ra1(instruction[24:20]),.rd0(ReadData1),.rd1(ReadData2),.wa(instruction[11:7]),.wd(RegWriteData),.we(RegWrite));
PC_Ctrl2 PC_Ctrl2(.instruction(instruction),.PC(PC),.Out(Out));
Imm_Gen Imm_Gen(.instruction(instruction),.Imm(Imm));
ALU_control
ALU_control(.ALUOp(ALUOp),.instruction({instruction[30],instruction[14:12]}),.ALU_sel(ALUControlSignal));
MUX MUXu1(.s0(PC_add4),.s1(PC_offseted),.sel(PCSrc),.d(Branched)); //mux upper 1st
MUX MUXu2(.s0(Branched),.s1(Out),.sel(Jal),.d(Jumped)); //mux upper 2nd
MUX MUXu3(.s0(Jumped),.s1(ALUResult),.sel(Jalr),.d(NPC)); //mux upper 3rd
MUX MUXd1(.s0(ReadData2),.s1(Imm),.sel(ALUSrc),.d(ALUIn2)); //mux down 1st
MUX MUXd2(.s0(ALUResult),.s1(ReadMemory),.sel(MemtoReg),.d(MUXd2Out)); //mux down 2nd
MUX MUXd3(.s0(MUXd2Out),.s1(PC_add4),.sel(Jal|Jalr),.d(MUXd3Out)); //mux down 3rd
MUX MUXd4(.s0(MUXd3Out),.s1(Out),.sel(Aquipc),.d(RegWriteData)); //mux down 4th
MUX5 MUXDebugReg(.s0(instruction[19:15]),.s1(chk_addr[4:0]),.sel(DebugReg),.d(RegAddr));
MUX MUXDebugMem(.s0(ALUResult),.s1({20'b0,chk_addr[11:0]}),.sel(DebugMem),.d(MemAddr));
data_memory
data_memory(.MemWrite(MemWrite),.MemRead(MemRead|DebugMem),.clk(clk),.Address(MemAddr),.WriteData(ReadData2),.io_din(io_din),.ReadData(ReadMemory),.io_dout(io_dout),.io_addr(io_addr),.io_we(io_we),.io_rd(io_rd));
PCSrc_driver
PCSrc_driver(.Branch(Branch),.Zero({instruction[14:12],ALUZero}),.PCSrc(PCSrc));
chk_data_driver
chk_data_driver(chk_addr,NPC,PC,instruction,ReadData1,ReadData2,Imm,ALUResult,ReadMemory,{Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite},chk_data);
 endmodule
```

仿真文件为不包括调试信号的CPU(和前面画的数据通路吻合)。

```
//test_CPU.v
`timescale 1ns / 1ps
module test_CPU();
reg clk,rstn;
wire[31:0]
PC,NPC,instruction,Out,PC_add4,Imm,PC_offseted,Jumped,Branched,ReadData1,ReadData2,RegWriteData,ReadMemory,ALUResult,ALUIn2,MUXd2Out,MUXd3Out;
wire Branch,MemRead,MemtoReg,MemWrite,ALUSrc,Jalr,Aquipc,Jal,RegWrite,PCSrc;
wire[1:0] ALUOp;
wire[2:0] ALUZero,ALUControlSignal;
PC_driver PC_driver(.NPC(NPC),.clk(clk),.rstn(rstn),.PC(PC));
instruction_memory instruction_memory(.clk(clk),.PC(PC),.instruction(instruction));
```

```
Add Add1(.a(PC),.b(32'h4),.y(PC_add4)); //Add PC by 4
Add Add2(.a(PC),.b(Imm<<1),.y(PC_offseted)); //Add PC by offset
ALU ALU(.a(ReadData1),.b(ALUIn2),.s(ALUControlSignal),.y(ALUResult),.f(ALUZero));
control
control(.Opcode(instruction[6:0]),.Branch(Branch),.MemRead(MemRead),.MemtoReg(MemtoReg),.Me
mWrite(MemWrite),.ALUSrc(ALUSrc),.Jalr(Jalr),.Aquipc(Aquipc),.Jal(Jal),.RegWrite(RegWrite),
.ALUOp(ALUOp));
registers
registers(.clk(clk),.ra0(instruction[19:15]),.ra1(instruction[24:20]),.rd0(ReadData1),.rd1(
ReadData2),.wa(instruction[11:7]),.wd(RegWriteData),.we(RegWrite));
PC_Ctrl2 PC_Ctrl2(.instruction(instruction),.PC(PC),.Out(Out));
Imm_Gen Imm_Gen(.instruction(instruction),.Imm(Imm));
ALU_control
ALU_control(.ALUOp(ALUOp),.instruction({instruction[30],instruction[14:12]}),.ALU_sel(ALUCo
ntrolSignal));
MUX MUXu1(.s0(PC_add4),.s1(PC_offseted),.sel(PCSrc),.d(Branched)); //mux upper 1st
MUX MUXu2(.s0(Branched),.s1(Out),.sel(Jal),.d(Jumped)); //mux upper 2nd
MUX MUXu3(.s0(Jumped),.s1(ALUResult),.sel(Jalr),.d(NPC)); //mux upper 3rd
MUX MUXd1(.s0(ReadData2),.s1(Imm),.sel(ALUSrc),.d(ALUIn2)); //mux down 1st
MUX MUXd2(.s0(ALUResult),.s1(ReadMemory),.sel(MemtoReg),.d(MUXd2Out)); //mux down 2nd
MUX MUXd3(.s0(MUXd2Out),.s1(PC_add4),.sel(Jal|Jalr),.d(MUXd3Out)); //mux down 3rd
MUX MUXd4(.s0(MUXd3Out),.s1(Out),.sel(Aquipc),.d(RegWriteData)); //mux down 4th
data_memory
data_memory(.MemWrite(MemWrite),.MemRead(MemRead),.clk(clk),.Address(ALUResult),.WriteData(
ReadData2),.ReadData(ReadMemory));
PCSrc_driver
PCSrc_driver(.Branch(Branch),.Zero({instruction[14:12],ALUZero}),.PCSrc(PCSrc));
initial
begin
    clk=0;
    rstn=0;
    #1 rstn=1'b1;
end
initial
begin
#4;
forever #1
clk=~clk;
end
initial
#200 $finish;
endmodule
```

PDU

这里直接复制老师的PDU模块。

```
//PDU.v
module  PDU(
        input clk,            //clk100mhz
        input rstn,           //cpu_resetn

        input step,           //btnu
        input cont,           //btnd
        input chk,            //btnr
        input data,           //btnc
        input del,            //btnl
```

```
    input [15:0] x,       //sw15-0

    output stop,          //led16r
    output [15:0] led,    //led15-0
    output [7:0] an,      //an7-0
    output [6:0] seg,     //ca-cg
    output [2:0] seg_sel, //led17

    output clk_cpu,       //cpu's clk
    output rst_cpu,       //cpu's rst

    //IO_BUS
    input [7:0] io_addr,
    input [31:0] io_dout,
    input io_we,
    input io_rd,
    output [31:0] io_din,

    //Debug_BUS
    input [31:0] chk_pc,    //连接CPU的npc
    output [15:0] chk_addr,
    input [31:0] chk_data
);

reg [15:0] rstn_r;
wire rst;               //复位信号，高电平有效
wire clk_pdu;           //PDU工作时钟
wire clk_db;            //去抖动计数器时钟
reg run_r, run_n;

reg [19:0] cnt_clk_r;   //时钟分频、数码管刷新计数器
reg [4:0] cnt_sw_db_r;
reg [15:0] x_db_r, x_db_1r;
reg xx_r, xx_1r;
wire x_p;
reg [3:0] x_hd_t;

wire [4:0] btn;
reg [4:0] cnt_btn_db_r;
reg [4:0] btn_db_r, btn_db_1r;
wire step_p, cont_p, chk_p, data_p, del_p;

reg [15:0] led_data_r;  //指示灯led15-0数据
reg [31:0] seg_data_r;  //数码管输出数据
reg seg_rdy_r;          //数码管准备好标志
reg [31:0] swx_data_r;  //开关输入数据
reg swx_vld_r;          //开关输入有效标志
reg [31:0] cnt_data_r;  //性能计数器数据

reg [31:0] tmp_r;       //临时编辑数据
reg [31:0] brk_addr_r;  //断点地址
reg [15:0] chk_addr_r;  //查看地址
reg [31:0] io_din_t;

reg led_sel_r;
reg [2:0] seg_sel_r;
reg [31:0] disp_data_t;
reg [7:0] an_t;
reg [3:0] hd_t;
```

```
    reg [6:0] seg_t;

    assign rst = rstn_r[15];    //经处理后的复位信号，高电平有效
    assign rst_cpu = rst;
    assign clk_pdu = cnt_clk_r[1];      //PDU工作时钟25MHz
    assign clk_db = cnt_clk_r[16];      //去抖动计数器时钟763Hz（周期约1.3ms）
    assign clk_cpu = clk_pdu & run_n;   //CPU工作时钟

    //降低指示灯和数码管亮度（保护眼睛）
    assign stop = ~run_r & (& cnt_clk_r[15:11]);
    assign led = ((led_sel_r)? chk_addr : led_data_r) & {{16{&cnt_clk_r[15:13]}}};
    assign an = an_t;
    assign seg = seg_t;
    assign seg_sel = seg_sel_r & {{3{&cnt_clk_r[15:11]}}};

    assign io_din = io_din_t;
    assign chk_addr = chk_addr_r;

    assign btn ={step, cont, chk, data, del};
    assign x_p = xx_r ^ xx_1r;
    assign step_p = btn_db_r[4] & ~ btn_db_1r[4];
    assign cont_p = btn_db_r[3] & ~ btn_db_1r[3];
    assign chk_p = btn_db_r[2] & ~ btn_db_1r[2];
    assign data_p = btn_db_r[1] & ~ btn_db_1r[1];
    assign del_p = btn_db_r[0] & ~ btn_db_1r[0];


    ///////////////////////////////////////////////
    //复位处理：异步复位、同步和延迟释放
    ///////////////////////////////////////////////
    always @(posedge clk, negedge rstn) begin
        if (!rstn)
            rstn_r <= 16'hFFFF;
        else
            rstn_r <= {rstn_r[14:0], 1'b0};
    end


    ///////////////////////////////////////////////
    //时钟分频
    ///////////////////////////////////////////////
    always @(posedge clk or posedge rst) begin
        if (rst)
            cnt_clk_r <= 20'h0;
        else
            cnt_clk_r <= cnt_clk_r + 20'h1;
    end


    ///////////////////////////////////////////////
    //开关sw去抖动
    ///////////////////////////////////////////////
    always @(posedge clk_db or posedge rst) begin
        if (rst)
            cnt_sw_db_r <= 5'h0;
        else if ((|(x ^ x_db_r)) & (~cnt_sw_db_r[4]))
            cnt_sw_db_r <= cnt_sw_db_r + 5'h1;
        else
            cnt_sw_db_r <= 5'h0;
    end

    always@(posedge clk_db or posedge rst) begin
        if (rst) begin
            x_db_r <= x;
            x_db_1r <= x;
            xx_r <= 1'b0;
        end
        else if (cnt_sw_db_r[4]) begin    //信号稳定约21ms后输出
            x_db_r <= x;
            x_db_1r <= x_db_r;
            xx_r <= ~xx_r;
        end
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            xx_1r <= 1'b0;
        else
            xx_1r <= xx_r;
    end


    ///////////////////////////////////////////////
    //按钮btn去抖动
    ///////////////////////////////////////////////
    always @(posedge clk_db or posedge rst) begin
        if (rst)
            cnt_btn_db_r <= 5'h0;
        else if ((|(btn ^ btn_db_r)) & (~cnt_btn_db_r[4]))
            cnt_btn_db_r <= cnt_btn_db_r + 5'h1;
        else
            cnt_btn_db_r <= 5'h0;
    end

    always@(posedge clk_db or posedge rst) begin
        if (rst)
            btn_db_r <= btn;
        else if (cnt_btn_db_r[4])
            btn_db_r <= btn;
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            btn_db_1r <= btn;
        else
            btn_db_1r <= btn_db_r;
    end


    ///////////////////////////////////////////////
    //控制CPU运行方式
    ///////////////////////////////////////////////
    reg [1:0] cs, ns;
    parameter STOP = 2'b00, STEP = 2'b01, RUN = 2'b10;

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            cs <= STOP;
        else
            cs <= ns;
    end

    always @* begin
        ns = cs;
        case (cs)
            STOP: begin
                if (step_p)
                    ns = STEP;
                else if (cont_p)
                    ns = RUN;
            end
            STEP:
                ns = STOP;
            RUN: begin
                if (brk_addr_r == chk_pc)
                    ns = STOP;
            end
            default:
                ns = STOP;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            run_r <= 1'b0;
        else if (ns == STOP)
            run_r <= 1'b0;
        else
            run_r <= 1'b1;
    end

    always @(negedge clk_pdu or posedge rst) begin
        if (rst)
            run_n <= 1'b0;
        else
            run_n <= run_r;
    end


    ///////////////////////////////////////////////
    //CPU输入/输出
    ///////////////////////////////////////////////
    always @(posedge clk_cpu or posedge rst) begin    //CPU输出    //fix
        if (rst) begin
            led_data_r <= 16'hFFFF;
            seg_data_r <= 32'h12345678;
        end
        else if (io_we) begin
            case (io_addr)
                8'h00:
                    led_data_r <= io_dout;
                8'h0C:
                    seg_data_r <= io_dout;
                default: ;
            endcase
        end
    end
    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            seg_rdy_r <= 1;
        else if (run_r & io_we & (io_addr == 8'h0C))    //fix
            seg_rdy_r <= 0;
        else if (x_p | del_p)
            seg_rdy_r <= 1;
    end

    always @(*) begin    //CPU输入
        case (io_addr)
            8'h04:
                io_din_t = {{11{1'b0}}, step, cont, chk, data, del, x};
            8'h08:
                io_din_t = {{31{1'b0}}, seg_rdy_r};
            8'h10:
                io_din_t = {{31{1'b0}}, swx_vld_r};
            8'h14:
                io_din_t = swx_data_r;
            8'h18:
                io_din_t = cnt_data_r;
            default:
                io_din_t = 32'h0;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            swx_vld_r <= 0;
        else if (data_p & ~swx_vld_r)
            swx_vld_r <= 1;
        else if (run_r & io_rd & (io_addr == 8'h14))    //fix
            swx_vld_r <= 0;
    end


    ///////////////////////////////////////////////
    //性能计数器
    ///////////////////////////////////////////////
    always@(posedge clk_cpu or posedge rst) begin
        if(rst)
            cnt_data_r <= 32'h0;
        else
            cnt_data_r <= cnt_data_r + 32'h1;
    end


    ///////////////////////////////////////////////
    //开关编辑数据
    ///////////////////////////////////////////////
    always @* begin    //开关输入编码
        case (x_db_r ^ x_db_1r )
            16'h0001:
                x_hd_t = 4'h0;
            16'h0002:
                x_hd_t = 4'h1;
            16'h0004:
                x_hd_t = 4'h2;
            16'h0008:
                x_hd_t = 4'h3;
            16'h0010:
                x_hd_t = 4'h4;
            16'h0020:
                x_hd_t = 4'h5;
            16'h0040:
                x_hd_t = 4'h6;
            16'h0080:
                x_hd_t = 4'h7;
            16'h0100:
                x_hd_t = 4'h8;
            16'h0200:
                x_hd_t = 4'h9;
            16'h0400:
                x_hd_t = 4'hA;
            16'h0800:
                x_hd_t = 4'hB;
            16'h1000:
                x_hd_t = 4'hC;
            16'h2000:
                x_hd_t = 4'hD;
            16'h4000:
                x_hd_t = 4'hE;
            16'h8000:
                x_hd_t = 4'hF;
            default:
                x_hd_t = 4'h0;
        endcase
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            tmp_r <= 32'h0;
        else if (x_p)
            tmp_r <= {tmp_r[27:0], x_hd_t};      //x_hd_t + tmp_r << 4
        else if (del_p)
            tmp_r <= {{4{1'b0}}, tmp_r[31:4]};   //tmp_r >> 4
        else if ((cont_p & ~run_r) | (data_p & ~swx_vld_r))
            tmp_r <= 32'h0;
        else if (chk_p & ~run_r)
            tmp_r <= tmp_r + 32'h1;
    end

    always @(posedge clk_pdu or posedge rst) begin
        if (rst) begin
            chk_addr_r <= 16'h0;
            brk_addr_r <= 32'h0;
        end
        else if (data_p & ~swx_vld_r)
            swx_data_r <= tmp_r;
        else if (cont_p & ~run_r)
            brk_addr_r <= tmp_r;
        else if (chk_p & ~run_r)
            chk_addr_r <= tmp_r;
    end


    ///////////////////////////////////////////////
    //led15-0指示灯显示
    ///////////////////////////////////////////////
    always @(posedge clk_pdu or posedge rst) begin
        if (rst)
            led_sel_r <= 1'b0;
        else if (run_r & io_we && (io_addr == 8'h00))    //fix
            led_sel_r <= 1'b0;
        else if (chk_p & ~run_r)
            led_sel_r <= 1'b1;
    end


    ///////////////////////////////////////////////
    //数码管多用途显示
    ///////////////////////////////////////////////
    always @(posedge clk_pdu or posedge rst) begin    //数码管显示数据选择
        if (rst)
            seg_sel_r <= 3'b001;
        else if (run_r & io_we & (io_addr == 8'h0C))    //fix
            seg_sel_r <= 3'b001;   //输出
        else if (x_p | del_p)
            seg_sel_r <= 3'b010;   //编辑
        else if (chk_p & ~run_r)
            seg_sel_r <= 3'b100;   //调试
    end

    always @* begin
        case (seg_sel_r)
            3'b001:
                disp_data_t = seg_data_r;
            3'b010:
                disp_data_t = tmp_r;
            3'b100:
                disp_data_t = chk_data;
            default:
                disp_data_t = tmp_r;
        endcase
    end

    always @(*) begin          //数码管扫描
        an_t <= 8'b1111_1111;
        hd_t <= disp_data_t[3:0];
        if (&cnt_clk_r[16:15])    //降低亮度
        case (cnt_clk_r[19:17])   //刷新频率约为95Hz
            3'b000: begin
                an_t <= 8'b1111_1110;
                hd_t <= disp_data_t[3:0];
            end
            3'b001: begin
                an_t <= 8'b1111_1101;
                hd_t <= disp_data_t[7:4];
            end
            3'b010: begin
                an_t <= 8'b1111_1011;
                hd_t <= disp_data_t[11:8];
            end
            3'b011: begin
                an_t <= 8'b1111_0111;
                hd_t <= disp_data_t[15:12];
            end
            3'b100: begin
                an_t <= 8'b1110_1111;
                hd_t <= disp_data_t[19:16];
            end
            3'b101: begin
                an_t <= 8'b1101_1111;
                hd_t <= disp_data_t[23:20];
            end
            3'b110: begin
                an_t <= 8'b1011_1111;
                hd_t <= disp_data_t[27:24];
            end
            3'b111: begin
                an_t <= 8'b0111_1111;
                hd_t <= disp_data_t[31:28];
            end
            default:
                ;
        endcase
    end

    always @ (*) begin    //7段译码
        case(hd_t)
            4'b1111:
                seg_t = 7'b0111000;
            4'b1110:
                seg_t = 7'b0110000;
            4'b1101:
                seg_t = 7'b1000010;
            4'b1100:
                seg_t = 7'b0110001;
            4'b1011:
                seg_t = 7'b1100000;
            4'b1010:
                seg_t = 7'b0001000;
            4'b1001:
                seg_t = 7'b0001100;
            4'b1000:
                seg_t = 7'b0000000;
            4'b0111:
                seg_t = 7'b0001111;
            4'b0110:
                seg_t = 7'b0100000;
            4'b0101:
                seg_t = 7'b0100100;
            4'b0100:
                seg_t = 7'b1001100;
            4'b0011:
                seg_t = 7'b0000110;
            4'b0010:
                seg_t = 7'b0010010;
            4'b0001:
                seg_t = 7'b1001111;
            4'b0000:
                seg_t = 7'b0000001;
            default:
                seg_t = 7'b1111111;
        endcase
```

```
    end

endmodule
```

top

这是主模块，往板子里烧的。

```
//top.v
`timescale 1ns / 1ps
module  top (
  input clk,    //clk100mhz
  input rstn,   //cpu_resetn

  input step,   //btnu
  input cont,   //btnd
  input chk,    //btnr
  input data,   //btnc
  input del,    //btnl
  input [15:0]  x,  //sw15-0

  output stop,      //led16r
  output [15:0] led,    //led15-0
  output [7:0] an,      //an7-0
  output [6:0] seg,     //ca-cg
  output [2:0] seg_sel); //led17
wire[7:0] io_addr;
wire[31:0] io_dout,io_din,chk_pc,chk_data;
wire io_we,io_rd,rst_cpu,clk_cpu;
wire[15:0] chk_addr;
PDU
PDU(.clk(clk),.rstn(rstn),.step(step),.cont(cont),.chk(chk),.data(data),.del(del),.x(x),.stop(stop),.led(led),.an(an),.seg(seg),.seg_sel(seg_sel),.clk_cpu(clk_cpu),
.rst_cpu(rst_cpu),.io_addr(io_addr),.io_dout(io_dout),.io_we(io_we),.io_rd(io_rd),.io_din(io_din),.chk_pc(chk_pc),.chk_addr(chk_addr),.chk_data(chk_data));
CPU CPU(clk_cpu,rst_cpu,io_addr,io_dout,io_we,io_rd,io_din,chk_pc,chk_addr,chk_data);
endmodule
```

其实就是简单地把CPU和PDU串起来。

运行

10条指令的测试程序如下：

```
.data
data0:.word 0xffff
data1:.word 0x001a
data2:.word 0x0001
test_add:.word 0x001b
test_addi:.word 0x001e
test_sub:.word 0x001d
test_auipc:.word 0x4068
.text
sw x0,0(x0) #Test sw
lw t0,0(x0) #Test lw
```

```
beq t0,x0,CONTINUE1 #Test beq
beq x0,x0,EXIT
CONTINUE1:
lw t0,4(x0)
blt x0,t0,CONTINUE2 #Test blt
beq x0,x0,EXIT
CONTINUE2:
lw t0,data1
lw t1,data2
add t0,t0,t1 #Test add
lw t2,test_add
beq t0,t2,PASS1
beq x0,x0,EXIT
PASS1:
addi t0,t0,0x0003 #Test addi
lw t2,test_addi
beq t0,t2,PASS2
beq x0,x0,EXIT
PASS2:
sub t0,t0,t1 #Test sub
lw t2,test_sub
beq t0,t2,PASS3
beq x0,x0,EXIT
PASS3:
auipc t0,0x0001 #Test auipc
lw t2,test_auipc
beq t0,t2,PASS4
beq x0,x0,EXIT
PASS4:
addi ra,x0,0
jal ra,CONTINUE3 #Test jal
addi s0,x0,1
beq x0,x0,EXIT
CONTINUE3:
addi t0,ra,-4
jalr ra,4(t0) #Test jalr
addi t0,x0,0xff
EXIT:
```

得到的test_data.coe：

```
memory_initialization_radix  = 16;
memory_initialization_vector =
0000ffff
0000001a
00000001
0000001b
0000001e
0000001d
00004068
```

得到的test_text.coe：

```
memory_initialization_radix  = 16;
memory_initialization_vector =
00002023
00002283
00028463
08000663
00402283
00504463
08000063
ffffd297
fe82a283
ffffd317
fe432303
006282b3
ffffd397
fdc3a383
00728463
04000e63
00328293
ffffd397
fcc3a383
00728463
04000463
406282b3
ffffd397
fbc3a383
00728463
02000a63
00001297
ffffd397
fac3a383
00728463
02000063
00000093
00c000ef
00100413
00000863
ffc08293
004280e7
0ff00293
```

排序的汇编做了稍许修改：

- 由于键盘输入会降低调试速度，所以干脆改成了内存中直接存放256个数。
- PDU的内存映射输出是写真实想输出的数字，而不是ASCII码，因此关于输出的部分可以省去很多。
- 输出的内存映射地址由0x7f08,0x7f0c变成了0xff08,0xff0c。

```
.data
array:.word 0x0030
mask:.word 0x0018
display_st:.word 0xff08
display_data:.word 0xff0c
mask1:.word 0x10000000
mask2:.word 0x01000000
mask3:.word 0x00100000
mask4:.word 0x00010000
mask5:.word 0x00001000
mask6:.word 0x00000100
mask7:.word 0x00000010
```

```
mask8:.word 0x00000001
.word 0xff
.word 0xfe
.word 0xfd
.word 0xfc
.word 0xfb
.word 0xfa
.word 0xf9
.word 0xf8
.word 0xf7
.word 0xf6
.word 0xf5
.word 0xf4
.word 0xf3
.word 0xf2
.word 0xf1
.word 0xf0
.word 0xef
.word 0xee
.word 0xed
.word 0xec
.word 0xeb
.word 0xea
.word 0xe9
.word 0xe8
.word 0xe7
.word 0xe6
.word 0xe5
.word 0xe4
.word 0xe3
.word 0xe2
.word 0xe1
.word 0xe0
.word 0xdf
.word 0xde
.word 0xdd
.word 0xdc
.word 0xdb
.word 0xda
.word 0xd9
.word 0xd8
.word 0xd7
.word 0xd6
.word 0xd5
.word 0xd4
.word 0xd3
.word 0xd2
.word 0xd1
.word 0xd0
.word 0xcf
.word 0xce
.word 0xcd
.word 0xcc
.word 0xcb
.word 0xca
.word 0xc9
.word 0xc8
.word 0xc7
.word 0xc6
```

```
    .word 0xc5
    .word 0xc4
    .word 0xc3
    .word 0xc2
    .word 0xc1
    .word 0xc0
    .word 0xbf
    .word 0xbe
    .word 0xbd
    .word 0xbc
    .word 0xbb
    .word 0xba
    .word 0xb9
    .word 0xb8
    .word 0xb7
    .word 0xb6
    .word 0xb5
    .word 0xb4
    .word 0xb3
    .word 0xb2
    .word 0xb1
    .word 0xb0
    .word 0xaf
    .word 0xae
    .word 0xad
    .word 0xac
    .word 0xab
    .word 0xaa
    .word 0xa9
    .word 0xa8
    .word 0xa7
    .word 0xa6
    .word 0xa5
    .word 0xa4
    .word 0xa3
    .word 0xa2
    .word 0xa1
    .word 0xa0
    .word 0x9f
    .word 0x9e
    .word 0x9d
    .word 0x9c
    .word 0x9b
    .word 0x9a
    .word 0x99
    .word 0x98
    .word 0x97
    .word 0x96
    .word 0x95
    .word 0x94
    .word 0x93
    .word 0x92
    .word 0x91
    .word 0x90
    .word 0x8f
    .word 0x8e
    .word 0x8d
    .word 0x8c
    .word 0x8b
    .word 0x8a
    .word 0x89
    .word 0x88
    .word 0x87
    .word 0x86
    .word 0x85
    .word 0x84
    .word 0x83
    .word 0x82
    .word 0x81
    .word 0x80
    .word 0x7f
    .word 0x7e
    .word 0x7d
    .word 0x7c
    .word 0x7b
    .word 0x7a
    .word 0x79
    .word 0x78
    .word 0x77
    .word 0x76
    .word 0x75
    .word 0x74
    .word 0x73
    .word 0x72
    .word 0x71
    .word 0x70
    .word 0x6f
    .word 0x6e
    .word 0x6d
    .word 0x6c
    .word 0x6b
    .word 0x6a
    .word 0x69
    .word 0x68
    .word 0x67
    .word 0x66
    .word 0x65
    .word 0x64
    .word 0x63
    .word 0x62
    .word 0x61
    .word 0x60
    .word 0x5f
    .word 0x5e
    .word 0x5d
    .word 0x5c
    .word 0x5b
    .word 0x5a
    .word 0x59
    .word 0x58
    .word 0x57
    .word 0x56
    .word 0x55
    .word 0x54
    .word 0x53
    .word 0x52
    .word 0x51
    .word 0x50
    .word 0x4f
    .word 0x4e
    .word 0x4d
    .word 0x4c
    .word 0x4b
    .word 0x4a
    .word 0x49
    .word 0x48
    .word 0x47
    .word 0x46
    .word 0x45
    .word 0x44
    .word 0x43
    .word 0x42
    .word 0x41
    .word 0x40
    .word 0x3f
    .word 0x3e
    .word 0x3d
    .word 0x3c
    .word 0x3b
    .word 0x3a
    .word 0x39
    .word 0x38
    .word 0x37
    .word 0x36
    .word 0x35
    .word 0x34
    .word 0x33
    .word 0x32
    .word 0x31
    .word 0x30
    .word 0x2f
    .word 0x2e
    .word 0x2d
    .word 0x2c
    .word 0x2b
    .word 0x2a
    .word 0x29
    .word 0x28
    .word 0x27
    .word 0x26
    .word 0x25
    .word 0x24
    .word 0x23
    .word 0x22
    .word 0x21
    .word 0x20
    .word 0x1f
    .word 0x1e
    .word 0x1d
    .word 0x1c
    .word 0x1b
    .word 0x1a
    .word 0x19
    .word 0x18
    .word 0x17
    .word 0x16
    .word 0x15
```

```
.word 0x14
.word 0x13
.word 0x12
.word 0x11
.word 0x10
.word 0x0f
.word 0x0e
.word 0x0d
.word 0x0c
.word 0x0b
.word 0x0a
.word 0x09
.word 0x08
.word 0x07
.word 0x06
.word 0x05
.word 0x04
.word 0x03
.word 0x02
.word 0x01
.word 0x00
.text
jal x0,main
#sort
sort: #a0 stores the base address,a1 stores the legnth of the array
addi t0,a0,0
add a1,a1,a1 #a1*=4
add a1,a1,a1
add t1,t0,a1
addi t2,t0,0 #i
OUTER_FOR:
beq t2,t1,END_OUTER
addi t3,t2,4 #j
INNER_FOR:
beq t3,t1,END_INNER
lw t4,(t2)
lw t5,(t3)
ble t4,t5,NEXT_INNER #no need to swap
addi t6,t4,0 #swap
addi t4,t5,0
addi t5,t6,0
sw t4,(t2) #restore
sw t5,(t3)
NEXT_INNER:
addi t3,t3,4 #j moves on
jal x0,INNER_FOR
END_INNER:
addi t2,t2,4 #i moves on
jal x0,OUTER_FOR
END_OUTER:
jalr x0,0(ra)
printf:
TRY:
lw t0,display_st
lw t0,(t0)
beq t0,x0,TRY
lw t0,display_data
sw a0,(t0)
jalr x0,0(ra)
```

```
#main
main:
addi s0,x0,0x0100 #Length
addi s1,x0,0
lw a0,array
addi a1,s0,0
jal ra,sort
lw s2,array #base address of the array
addi s1,x0,0 #How many printed
PRINT_ARRAY:
beq s1,s0,END_PRINT_ARRAY
lw a0,(s2)
jal ra,printf
addi s1,s1,1
addi s2,s2,4
jal x0,PRINT_ARRAY
END_PRINT_ARRAY:
EXIT:
```

得到的sort_data.coe：

```
memory_initialization_radix  = 16;
memory_initialization_vector =
00000030
00000018
0000ff08
0000ff0c
10000000
01000000
00100000
00010000
00001000
00000100
00000010
00000001
000000ff
000000fe
000000fd
000000fc
000000fb
000000fa
000000f9
000000f8
000000f7
000000f6
000000f5
000000f4
000000f3
000000f2
000000f1
000000f0
000000ef
000000ee
000000ed
000000ec
000000eb
000000ea
```

```
000000e9
000000e8
000000e7
000000e6
000000e5
000000e4
000000e3
000000e2
000000e1
000000e0
000000df
000000de
000000dd
000000dc
000000db
000000da
000000d9
000000d8
000000d7
000000d6
000000d5
000000d4
000000d3
000000d2
000000d1
000000d0
000000cf
000000ce
000000cd
000000cc
000000cb
000000ca
000000c9
000000c8
000000c7
000000c6
000000c5
000000c4
000000c3
000000c2
000000c1
000000c0
000000bf
000000be
000000bd
000000bc
000000bb
000000ba
000000b9
000000b8
000000b7
000000b6
000000b5
000000b4
000000b3
000000b2
000000b1
000000b0
000000af
```

```
000000ae
000000ad
000000ac
000000ab
000000aa
000000a9
000000a8
000000a7
000000a6
000000a5
000000a4
000000a3
000000a2
000000a1
000000a0
0000009f
0000009e
0000009d
0000009c
0000009b
0000009a
00000099
00000098
00000097
00000096
00000095
00000094
00000093
00000092
00000091
00000090
0000008f
0000008e
0000008d
0000008c
0000008b
0000008a
00000089
00000088
00000087
00000086
00000085
00000084
00000083
00000082
00000081
00000080
0000007f
0000007e
0000007d
0000007c
0000007b
0000007a
00000079
00000078
00000077
00000076
00000075
00000074
```

```
00000073
00000072
00000071
00000070
0000006f
0000006e
0000006d
0000006c
0000006b
0000006a
00000069
00000068
00000067
00000066
00000065
00000064
00000063
00000062
00000061
00000060
0000005f
0000005e
0000005d
0000005c
0000005b
0000005a
00000059
00000058
00000057
00000056
00000055
00000054
00000053
00000052
00000051
00000050
0000004f
0000004e
0000004d
0000004c
0000004b
0000004a
00000049
00000048
00000047
00000046
00000045
00000044
00000043
00000042
00000041
00000040
0000003f
0000003e
0000003d
0000003c
0000003b
0000003a
00000039
```

```
00000038
00000037
00000036
00000035
00000034
00000033
00000032
00000031
00000030
0000002f
0000002e
0000002d
0000002c
0000002b
0000002a
00000029
00000028
00000027
00000026
00000025
00000024
00000023
00000022
00000021
00000020
0000001f
0000001e
0000001d
0000001c
0000001b
0000001a
00000019
00000018
00000017
00000016
00000015
00000014
00000013
00000012
00000011
00000010
0000000f
0000000e
0000000d
0000000c
0000000b
0000000a
00000009
00000008
00000007
00000006
00000005
00000004
00000003
00000002
00000001
00000000
```

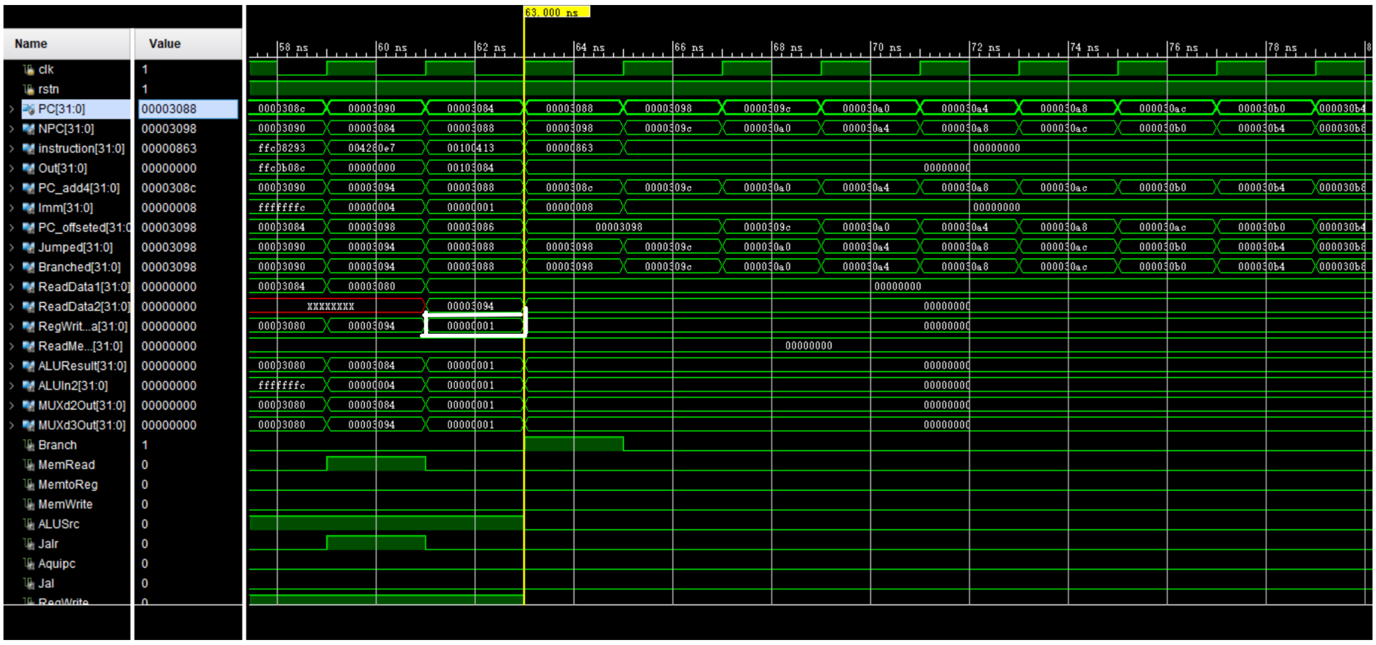
得到的sort_text.coe：

```
memory_initialization_radix  = 16;
memory_initialization_vector =
0780006f
00050293
00b585b3
00b585b3
00b28333
00028393
02638e63
00438e13
026e0663
0003ae83
000e2f03
01df5c63
000e8f93
000f0e93
000f8f13
01d3a023
01ee2023
004e0e13
fd9ff06f
00438393
fc9ff06f
00008067
ffffd297
fb02a283
0002a283
fe028ae3
ffffd297
fa42a283
00a2a023
00008067
10000413
00000493
ffffd517
f8052503
00040593
f79ff0ef
ffffd917
f7092903
00000493
00848c63
00092503
fb5ff0ef
00148493
00490913
fedff06f
```

仿真

10条指令的测试

载入test_data.coe和test_text.coe，如果测试成功，寄存器x8会被置为1，如图所示，测试成功。



排序测试

载入sort_data.coe和sort_text.coe，简要检查数据存储器中读出的值并跟踪验证数据交换即可。本过程从略。

上板测试

constr.xdc：

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level
signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];


##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { x[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { x[1] }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { x[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { x[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { x[4] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { x[5] }];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { x[6] }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { x[7] }];
```

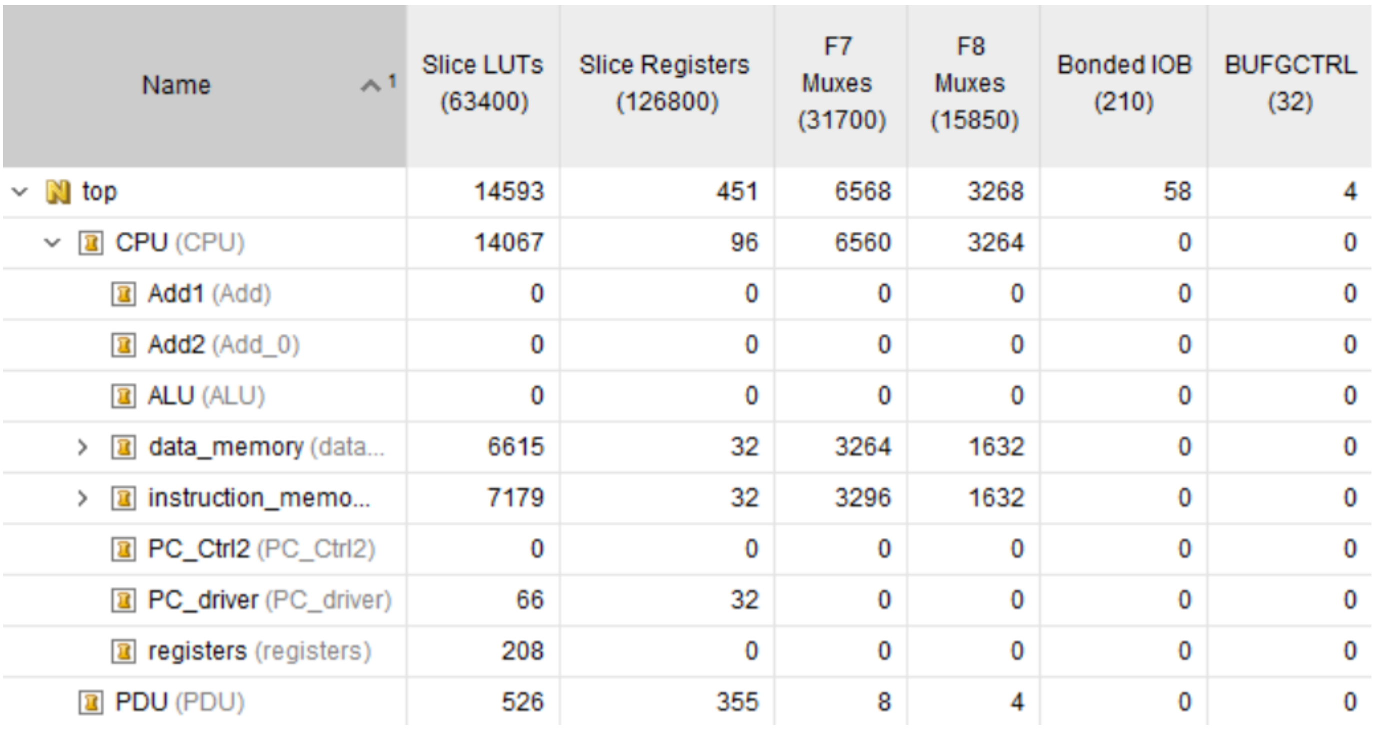
```
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { x[8] }];
#IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { x[9] }];
#IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { x[10] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { x[11] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { x[12] }];
#IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { x[13] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { x[14] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { x[15] }];
#IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { led[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { led[9] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { led[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { led[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { led[12] }];
#IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { led[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { led[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { led[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }];
#IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }];
#IO_L10P_T1_D14_14 Sch=led16_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { stop }];
```

```
#IO_L11P_T1_SRCC_14 Sch=led16_r
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { seg_sel[0] }];
#IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { seg_sel[1] }];
#IO_0_14 Sch=led17_g
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { seg_sel[2] }];
#IO_L11N_T1_SRCC_14 Sch=led17_r


##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[6] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[5] }];
#IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[4] }];
#IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3] }];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[2] }];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[1] }];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[0] }];
#IO_L4P_T0_D04_14 Sch=cg

#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }];
#IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { an[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { an[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { an[6] }];
#IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { an[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons

set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { rstn }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { data }];
#IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { step }];
#IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { del }];
#IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { chk }];
```

```
#IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { cont }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

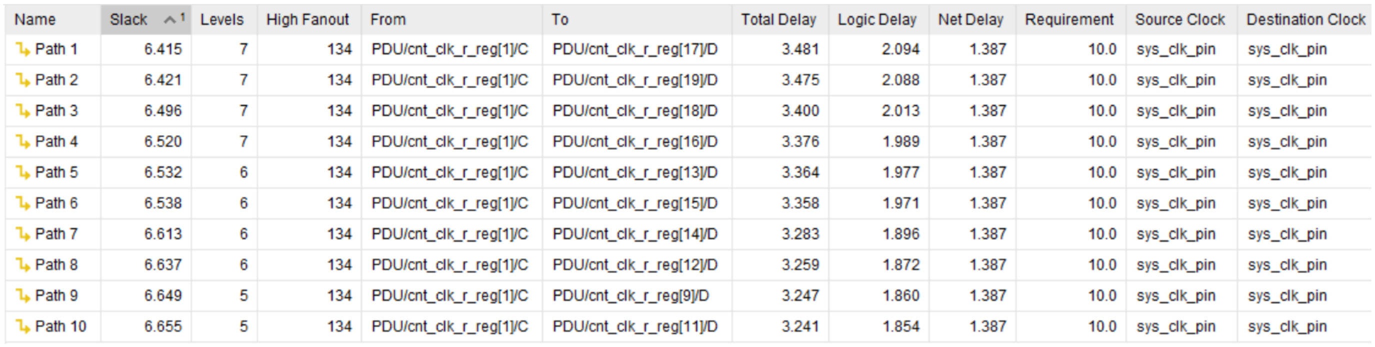
- 首先是10条指令：拨动开关输入3098（EXIT地址）设置断点，按cont，输入1008（查看8号寄存器），数码管显示00000001则表示测试通过。
- 然后是排序。直接按cont，然后数码管会显示8个0，这是升序排完序后数组的第一个元素。随后每按4次del或拨动四次开关显示下一个数字（四次是因为存储器中每两个数据隔4个字节），发现数字依次递增。
- 其他调试：到达断点后，输入chk_addr的值并按下chk，数码管会显示相应的chk_data，再按一下chk，数码管上会变成chk_addr+1对应的chk_data，或者按step，数码管上会变成下一条指令对应的PC，再按step则再下一条指令。

电路资源使用情况和电路性能



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
top	14593	451	6568	3268	58	4
CPU (CPU)	14067	96	6560	3264	0	0
Add1 (Add)	0	0	0	0	0	0
Add2 (Add_0)	0	0	0	0	0	0
ALU (ALU)	0	0	0	0	0	0
data_memory (data...	6615	32	3264	1632	0	0
instruction_memo...	7179	32	3296	1632	0	0
PC_Ctrl2 (PC_Ctrl2)	0	0	0	0	0	0
PC_driver (PC_driver)	66	32	0	0	0	0
registers (registers)	208	0	0	0	0	0
PDU (PDU)	526	355	8	4	0	0

可以看到LUT和Mux使用较多，这和两个存储器有很大关系：这两个资源大部分集中到了这两个存储器中。



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	6.415	7	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[17]/D	3.481	2.094	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 2	6.421	7	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[19]/D	3.475	2.088	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 3	6.496	7	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[18]/D	3.400	2.013	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 4	6.520	7	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[16]/D	3.376	1.989	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 5	6.532	6	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[13]/D	3.364	1.977	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 6	6.538	6	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[15]/D	3.358	1.971	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 7	6.613	6	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[14]/D	3.283	1.896	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 8	6.637	6	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[12]/D	3.259	1.872	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 9	6.649	5	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[9]/D	3.247	1.860	1.387	10.0	sys_clk_pin	sys_clk_pin
Path 10	6.655	5	134	PDU/cnt_clk_r_reg[1]/C	PDU/cnt_clk_r_reg[11]/D	3.241	1.854	1.387	10.0	sys_clk_pin	sys_clk_pin

数据通路延迟可以从表中看出，不是很大。

遇到的困难和解决方法

- 遇到combinational loop警告：报错信息里包含了回路属于哪个模块，但这并不准确，而且还有一个问题是它的回路是分析后的，基本得不到什么信息。最后的解决办法是用二分法逐步找到出错的指令，然后看这条指令中谁的值是有谁决定的，如果组合逻辑中B的值和A有关，那么思考是不是有一些过程把A赋给了B。如果找到这样一个过程，那A和B就是造成回路的原因。如果数据通路设计无误，大概率是变量名写错了。