## 3.2 实验六 20条指令的单周期CPU系统设计实验

### 1 实验目的

1. **了解龙芯LoongArch指令系统架构，掌握常用指令的功能和编码。**
2. **熟悉并掌握20条指令的单周期CPU的原理和设计。**
3. **加强运用Verilog语言进行电路设计的能力。**
4. **了解并使用基于trace比对的调试框架进行错误调试,具备初步调试能力。**

### 2 实验设备

Fbg676,LS-CPU-EXB,装有 vivado 环境的电脑

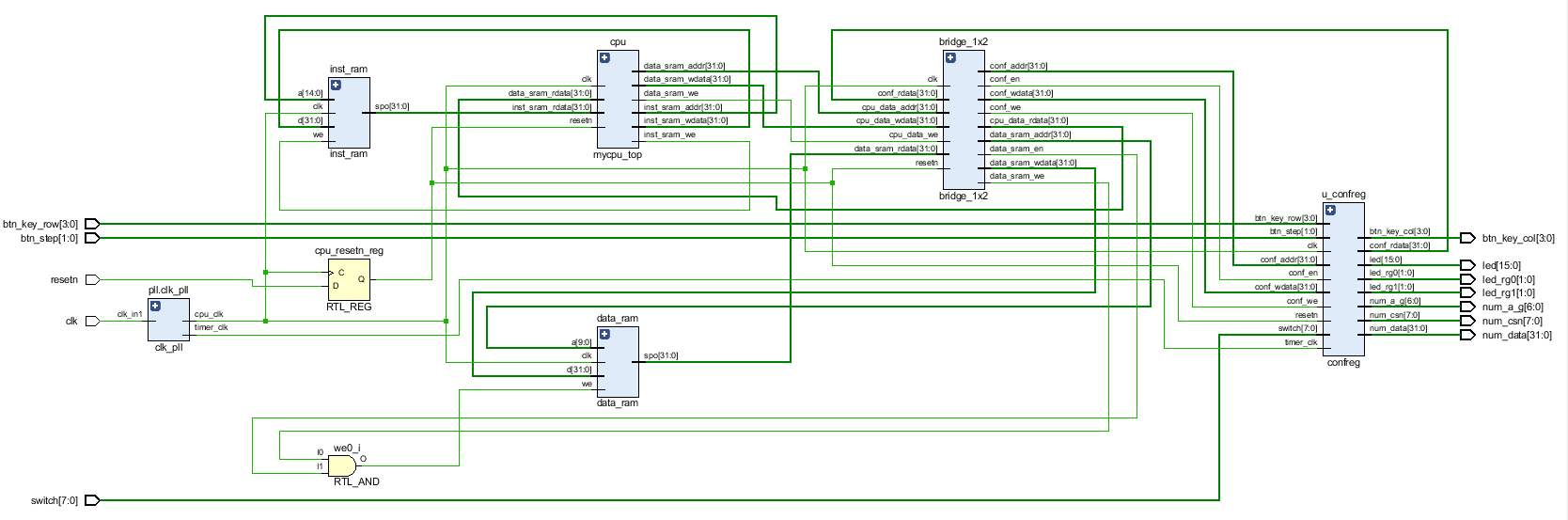
### 3 单周期CPU实现的20条LoongArch指令

表3.4 20条基础指令特性归纳表

|  |  |  |
| --- | --- | --- |
| 汇编指令 | 指令码 | 功能描述 |
| add.w rd, rj, rk | 00000000000100000|rk|rj|rd | GR[rd] = GR[rj]+GR[rk] |
| addi.w rd, rj, si12 | 0000001010|si12|rj|rd | GR[rd] = GR[rj] + SignExtend(si12, 32) |
| sub.w |  |  |
| ld.w rd,rj,si12 | 0010100010|si12|rj|rd | vaddr = GR[rj] + SignExtend(si12, 32) AddressComplianceCheck(vaddr) paddr = AddressTranslation(vaddr) word = MemoryLoad(paddr, WORD) GR[rd] = word |
| st.w rd,rj,si12 | 0010100110|si12|rj|rd | vaddr = GR[rj] + SignExtend(si12, 32) AddressComplianceCheck(vaddr) paddr = AddressTranslation(vaddr) MemoryStore(GR[rd][31:0], paddr, WORD) |
| beq rj, rd, offs | 010110| 0 offs[15:0]|rj|rd | if GR[rj]==GR[rd] : PC = PC + SignExtend({offs16, 2'b0}, 32) |
| bne rj,rd,offs | 010111|offs[15:0]|rj|rd | if GR[rj]!=GR[rd] : PC = PC + SignExtend({offs16, 2'b0}, 32) |
| b offs | 010100|offs[15:0]|offs[25:16] | PC = PC + SignExtend({offs26, 2'b0}, 32) |
| Bl offs | 010101|offs[15:0]|offs[25:16] | GR[1] = PC + 4 PC = PC + SignExtend({offs26, 2'b0}, 32) |
| Jirl rd, rj, offs | 010011|offs[15:0]|rj|rd | GR[rd] = PC + 4 PC = GR[rj] + SignExtend({offs16, 2'b0}, 32) |
| Slt rd,rj,rk | 00000000000100100|rk|rj|rd | GR[rd] = (signed(GR[rj]) <signed(GR[rk])) ? 1 : 0 |
| Sltu rd, rj, rk | 00000000000100101|rk|rj|rd | GR[rd] = (unsigned(GR[rj]) < unsigned(GR[rk])) ? 1 : 0 |
| Slli.w rd, rj, ui5 | 00000000010000|001|ui5|rj|rd | tmp = SLL(GR[rj], ui5) GR[rd] = tmp[31:0] |
| Srli.w rd, rj, ui5 | 00000000010001|001|ui5|rj|rd | tmp = SRL(GR[rj], ui5) GR[rd] = tmp[31:0] |
| Srai.w rd, rj, ui5 | 00000000010010|001|ui5|rj|rd | tmp = SRA(GR[rj], ui5) GR[rd] = tmp[31:0] |
| Lu12i.w rd, si20 | 0001010|si20|rd | GR[rd] = {si20, 12'b0} |
| And rd, rj, rk | 00000000000101001|rk|rj|rd | GR[rd] = GR[rj] & GR[rk] |
| Nor rd, rj, rk | 00000000000101000|rk|rj|rd | GR[rd] = ~(GR[rj] | GR[rk]) |
| Or rd, rj, rk | 00000000000101010|rk|rj|rd | GR[rd] = GR[rj] | GR[rk] |
| Xor rd, rj, rk | 00000000000101011|rk|rj|rd | GR[rd] = GR[rj] ^ GR[rk] |

### 4 实验内容与步骤

1. **画出模块框图，标出输入输出端口。**

****

1. **启动验证myCPU的工程，查看并修改设计文件，运行仿真，进行功能验证与调试，填写调试实验数据表3.5。**
2. **修改后的主要设计代码及注释（修改的代码请标红）。**

① mycpu\_top.v

module mycpu\_top(

input wire clk,

input wire resetn,

// inst sram interface

output wire inst\_sram\_we,

output wire [31:0] inst\_sram\_addr,

output wire [31:0] inst\_sram\_wdata,

input wire [31:0] inst\_sram\_rdata,

// data sram interface

output wire data\_sram\_we,

output wire [31:0] data\_sram\_addr,

output wire [31:0] data\_sram\_wdata,

input wire [31:0] data\_sram\_rdata,

// trace debug interface

output wire [31:0] debug\_wb\_pc,

output wire [ 3:0] debug\_wb\_rf\_we,

output wire [ 4:0] debug\_wb\_rf\_wnum,

output wire [31:0] debug\_wb\_rf\_wdata

);

reg reset;

always @(posedge clk) reset <= ~resetn;

reg valid;

always @(posedge clk) begin

if (reset) begin

valid <= 1'b0;

end

else begin

valid <= 1'b1;

end

end

wire [31:0] seq\_pc;

wire [31:0] nextpc;

wire br\_taken;

wire [31:0] br\_target;

wire [31:0] inst;

reg [31:0] pc;

wire [11:0] alu\_op;

wire load\_op;

wire src1\_is\_pc;

wire src2\_is\_imm;

wire res\_from\_mem;

wire dst\_is\_r1;

wire gr\_we;

wire mem\_we;

wire src\_reg\_is\_rd;

wire [4: 0] dest;

wire [31:0] rj\_value;

wire [31:0] rkd\_value;

wire [31:0] imm;

wire [31:0] br\_offs;

wire [31:0] jirl\_offs;

wire [ 5:0] op\_31\_26;

wire [ 3:0] op\_25\_22;

wire [ 1:0] op\_21\_20;

wire [ 4:0] op\_19\_15;

wire [ 4:0] rd;

wire [ 4:0] rj;

wire [ 4:0] rk;

wire [11:0] i12;

wire [19:0] i20;

wire [15:0] i16;

wire [25:0] i26;

wire [63:0] op\_31\_26\_d;

wire [15:0] op\_25\_22\_d;

wire [ 3:0] op\_21\_20\_d;

wire [31:0] op\_19\_15\_d;

wire inst\_add\_w;

wire inst\_sub\_w;

wire inst\_slt;

wire inst\_sltu;

wire inst\_nor;

wire inst\_and;

wire inst\_or;

wire inst\_xor;

wire inst\_slli\_w;

wire inst\_srli\_w;

wire inst\_srai\_w;

wire inst\_addi\_w;

wire inst\_ld\_w;

wire inst\_st\_w;

wire inst\_jirl;

wire inst\_b;

wire inst\_bl;

wire inst\_beq;

wire inst\_bne;

wire inst\_lu12i\_w;

wire need\_ui5;

wire need\_si12;

wire need\_si16;

wire need\_si20;

wire need\_si26;

wire src2\_is\_4;

wire [ 4:0] rf\_raddr1;

wire [31:0] rf\_rdata1;

wire [ 4:0] rf\_raddr2;

wire [31:0] rf\_rdata2;

wire rf\_we ;

wire [ 4:0] rf\_waddr;

wire [31:0] rf\_wdata;

wire [31:0] alu\_src1 ;

wire [31:0] alu\_src2 ;

wire [31:0] alu\_result ;

wire [31:0] mem\_result;

wire [31:0] final\_result;//添加final\_result

assign seq\_pc = pc + 3'h4;

assign nextpc = br\_taken ? br\_target : seq\_pc;

always @(posedge clk) begin

if (reset) begin

pc <= 32'h1bfffffc; //trick: to make nextpc be 0x1c000000 during reset

end

else begin

pc <= nextpc;

end

end

assign inst\_sram\_we = 1'b0;

assign inst\_sram\_addr = pc;

assign inst\_sram\_wdata = 32'b0;

assign inst = inst\_sram\_rdata;

assign op\_31\_26 = inst[31:26];

assign op\_25\_22 = inst[25:22];

assign op\_21\_20 = inst[21:20];

assign op\_19\_15 = inst[19:15];

assign rd = inst[ 4: 0];

assign rj = inst[ 9: 5];

assign rk = inst[14:10];

assign i12 = inst[21:10];

assign i20 = inst[24: 5];

assign i16 = inst[25:10];

assign i26 = {inst[ 9: 0], inst[25:10]};

decoder\_6\_64 u\_dec0(.in(op\_31\_26 ), .out(op\_31\_26\_d ));

decoder\_4\_16 u\_dec1(.in(op\_25\_22 ), .out(op\_25\_22\_d ));

decoder\_2\_4 u\_dec2(.in(op\_21\_20 ), .out(op\_21\_20\_d ));

decoder\_5\_32 u\_dec3(.in(op\_19\_15 ), .out(op\_19\_15\_d ));

assign inst\_add\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h00];

assign inst\_sub\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h02];

assign inst\_slt = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h04];

assign inst\_sltu = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h05];

assign inst\_nor = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h08];

assign inst\_and = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h09];

assign inst\_or = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h0a];

assign inst\_xor = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h0b];

assign inst\_slli\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h1] & op\_21\_20\_d[2'h0] & op\_19\_15\_d[5'h01];

assign inst\_srli\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h1] & op\_21\_20\_d[2'h0] & op\_19\_15\_d[5'h09];

assign inst\_srai\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h1] & op\_21\_20\_d[2'h0] & op\_19\_15\_d[5'h11];

assign inst\_addi\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'ha];

assign inst\_ld\_w = op\_31\_26\_d[6'h0a] & op\_25\_22\_d[4'h2];

assign inst\_st\_w = op\_31\_26\_d[6'h0a] & op\_25\_22\_d[4'h6];

assign inst\_jirl = op\_31\_26\_d[6'h13];

assign inst\_b = op\_31\_26\_d[6'h14];

assign inst\_bl = op\_31\_26\_d[6'h15];

assign inst\_beq = op\_31\_26\_d[6'h16];

assign inst\_bne = op\_31\_26\_d[6'h17];

assign inst\_lu12i\_w= op\_31\_26\_d[6'h05] & ~inst[25];

assign alu\_op[ 0] = inst\_add\_w | inst\_addi\_w | inst\_ld\_w | inst\_st\_w

| inst\_jirl | inst\_bl;

assign alu\_op[ 1] = inst\_sub\_w;

assign alu\_op[ 2] = inst\_slt;

assign alu\_op[ 3] = inst\_sltu;

assign alu\_op[ 4] = inst\_and;

assign alu\_op[ 5] = inst\_nor;

assign alu\_op[ 6] = inst\_or;

assign alu\_op[ 7] = inst\_xor;

assign alu\_op[ 8] = inst\_slli\_w;

assign alu\_op[ 9] = inst\_srli\_w;

assign alu\_op[10] = inst\_srai\_w;

assign alu\_op[11] = inst\_lu12i\_w;

assign need\_ui5 = inst\_slli\_w | inst\_srli\_w | inst\_srai\_w;

assign need\_si12 = inst\_addi\_w | inst\_ld\_w | inst\_st\_w;

assign need\_si16 = inst\_jirl | inst\_beq | inst\_bne;

assign need\_si20 = inst\_lu12i\_w;

assign need\_si26 = inst\_b | inst\_bl;

assign src2\_is\_4 = inst\_jirl | inst\_bl;

assign imm = src2\_is\_4 ? 32'h4 :

need\_si20 ? {i20[19:0], 12'b0} :

/\*need\_ui5 || need\_si12\*/{{20{i12[11]}}, i12[11:0]} ;

assign br\_offs = need\_si26 ? {{ 4{i26[25]}}, i26[25:0], 2'b0} :

{{14{i16[15]}}, i16[15:0], 2'b0} ;

assign jirl\_offs = {{14{i16[15]}}, i16[15:0], 2'b0};

assign src\_reg\_is\_rd = inst\_beq | inst\_bne | inst\_st\_w;

assign src1\_is\_pc = inst\_jirl | inst\_bl;

assign src2\_is\_imm = inst\_slli\_w |

inst\_srli\_w |

inst\_srai\_w |

inst\_addi\_w |

inst\_ld\_w |

inst\_st\_w |

inst\_lu12i\_w|

inst\_jirl |

inst\_bl ;

assign res\_from\_mem = inst\_ld\_w;

assign dst\_is\_r1 = inst\_bl;

assign gr\_we = ~inst\_st\_w & ~inst\_beq & ~inst\_bne & ~inst\_b;//删除& ~inst\_bl

assign mem\_we = inst\_st\_w;

assign dest = dst\_is\_r1 ? 5'd1 : rd;

assign rf\_raddr1 = rj;

assign rf\_raddr2 = src\_reg\_is\_rd ? rd :rk;

regfile u\_regfile(

.clk (clk ),

.raddr1 (rf\_raddr1),

.rdata1 (rf\_rdata1),

.raddr2 (rf\_raddr2),

.rdata2 (rf\_rdata2),

.we (rf\_we ),

.waddr (rf\_waddr ),

.wdata (rf\_wdata )

);

assign rj\_value = rf\_rdata1;

assign rkd\_value = rf\_rdata2;

assign rj\_eq\_rd = (rj\_value == rkd\_value);

assign br\_taken = ( inst\_beq && rj\_eq\_rd

|| inst\_bne && !rj\_eq\_rd

|| inst\_jirl

|| inst\_bl

|| inst\_b

) && valid;

assign br\_target = (inst\_beq || inst\_bne || inst\_bl || inst\_b) ? (pc + br\_offs) :

/\*inst\_jirl\*/ (rj\_value + jirl\_offs);

assign alu\_src1 = src1\_is\_pc ? pc[31:0] : rj\_value;

assign alu\_src2 = src2\_is\_imm ? imm : rkd\_value;

alu u\_alu(

.alu\_op (alu\_op ),

.alu\_src1 (alu\_src1 ),//修改成对应引脚

.alu\_src2 (alu\_src2 ),

.alu\_result (alu\_result)

);

assign data\_sram\_we = mem\_we && valid;

assign data\_sram\_addr = alu\_result;

assign data\_sram\_wdata = rkd\_value;

assign mem\_result = data\_sram\_rdata;

assign final\_result = res\_from\_mem ? mem\_result : alu\_result;

assign rf\_we = gr\_we && valid;

assign rf\_waddr = dest;

assign rf\_wdata = final\_result;

// debug info generate

assign debug\_wb\_pc = pc;

assign debug\_wb\_rf\_we = {4{rf\_we}};//变量名从debug\_wb\_rf\_wen改为debug\_wb\_rf\_we

assign debug\_wb\_rf\_wnum = dest;

assign debug\_wb\_rf\_wdata = final\_result;

endmodule

② alu.v

module alu(

input wire [11:0] alu\_op,

input wire [31:0] alu\_src1,

input wire [31:0] alu\_src2,

output wire [31:0] alu\_result

);

wire op\_add; //add operation

wire op\_sub; //sub operation

wire op\_slt; //signed compared and set less than

wire op\_sltu; //unsigned compared and set less than

wire op\_and; //bitwise and

wire op\_nor; //bitwise nor

wire op\_or; //bitwise or

wire op\_xor; //bitwise xor

wire op\_sll; //logic left shift

wire op\_srl; //logic right shift

wire op\_sra; //arithmetic right shift

wire op\_lui; //Load Upper Immediate

// control code decomposition

assign op\_add = alu\_op[ 0];

assign op\_sub = alu\_op[ 1];

assign op\_slt = alu\_op[ 2];

assign op\_sltu = alu\_op[ 3];

assign op\_and = alu\_op[ 4];

assign op\_nor = alu\_op[ 5];

assign op\_or = alu\_op[ 6];

assign op\_xor = alu\_op[ 7];

assign op\_sll = alu\_op[ 8];

assign op\_srl = alu\_op[ 9];

assign op\_sra = alu\_op[10];

assign op\_lui = alu\_op[11];

wire [31:0] add\_sub\_result;

wire [31:0] slt\_result;

wire [31:0] sltu\_result;

wire [31:0] and\_result;

wire [31:0] nor\_result;

wire [31:0] or\_result;

wire [31:0] xor\_result;

wire [31:0] lui\_result;

wire [31:0] sll\_result;

wire [63:0] sr64\_result;

wire [31:0] sr\_result;

// 32-bit adder

wire [31:0] adder\_a;

wire [31:0] adder\_b;

wire adder\_cin;

wire [31:0] adder\_result;

wire adder\_cout;

assign adder\_a = alu\_src1;

assign adder\_b = (op\_sub | op\_slt | op\_sltu) ? ~alu\_src2 : alu\_src2; //src1 - src2 rj-rk

assign adder\_cin = (op\_sub | op\_slt | op\_sltu) ? 1'b1 : 1'b0;

assign {adder\_cout, adder\_result} = adder\_a + adder\_b + adder\_cin;

// ADD, SUB result

assign add\_sub\_result = adder\_result;

// SLT result

assign slt\_result[31:1] = 31'b0; //rj < rk 1

assign slt\_result[0] = (alu\_src1[31] & ~alu\_src2[31])

| ((alu\_src1[31] ~^ alu\_src2[31]) & adder\_result[31]);

// SLTU result

assign sltu\_result[31:1] = 31'b0;

assign sltu\_result[0] = ~adder\_cout;

// bitwise operation

assign and\_result = alu\_src1 & alu\_src2;

assign or\_result = alu\_src1 | alu\_src2;//删除alu\_result

assign nor\_result = ~or\_result;

assign xor\_result = alu\_src1 ^ alu\_src2;

assign lui\_result = alu\_src2;

// SLL result

assign sll\_result = alu\_src1 << alu\_src2[4:0]; //修改成正确代码

// SRL, SRA result

assign sr64\_result = {{32{op\_sra & alu\_src1[31]}}, alu\_src1[31:0]} >> alu\_src2[4:0]; //src1和src2弄反了

assign sr\_result = sr64\_result[31:0];//改为32位

// final result mux

assign alu\_result = ({32{op\_add|op\_sub}} & add\_sub\_result)

| ({32{op\_slt }} & slt\_result)

| ({32{op\_sltu }} & sltu\_result)

| ({32{op\_and }} & and\_result)

| ({32{op\_nor }} & nor\_result)

| ({32{op\_or }} & or\_result)

| ({32{op\_xor }} & xor\_result)

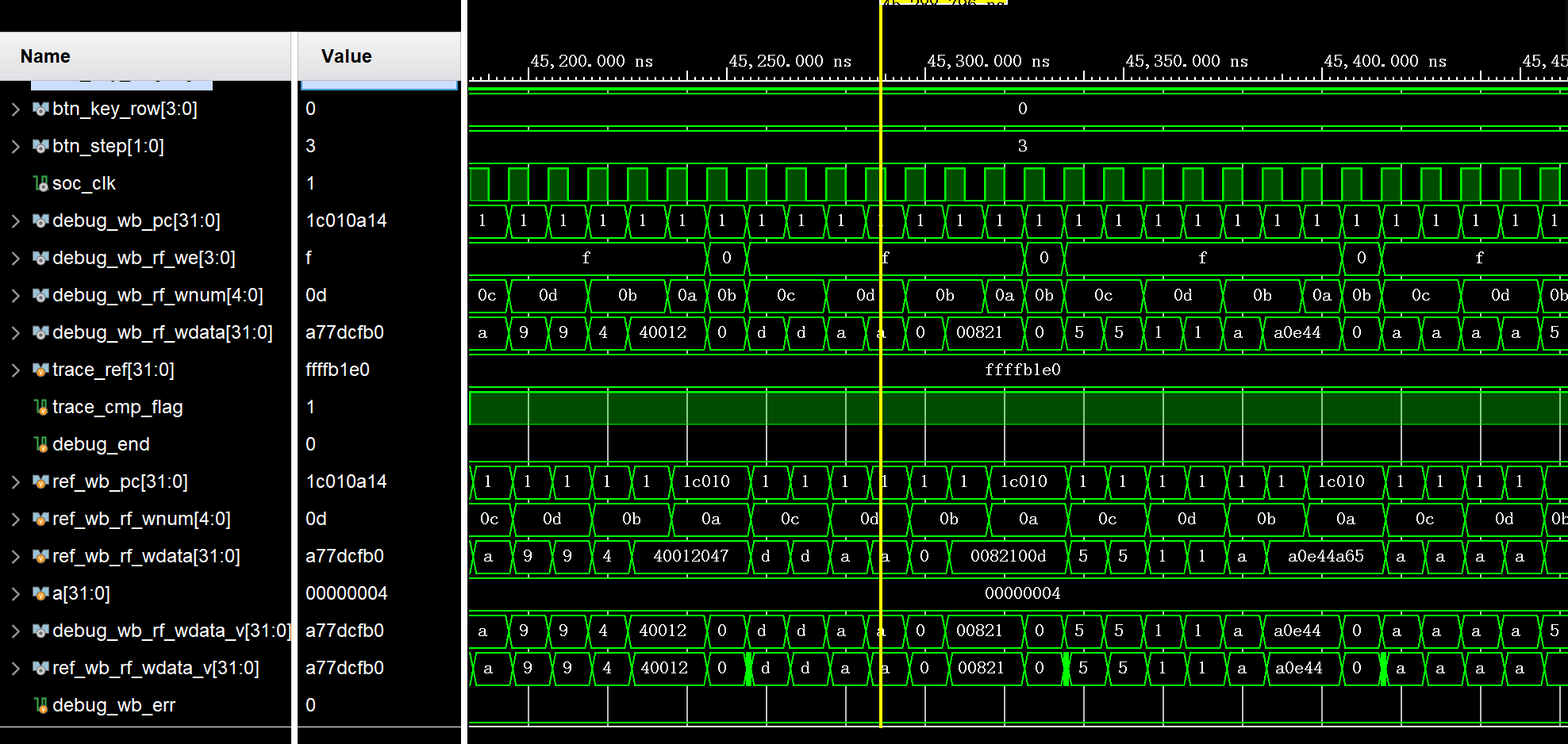
| ({32{op\_lui }} & lui\_result)

| ({32{op\_sll }} & sll\_result)

| ({32{op\_srl|op\_sra}} & sr\_result);

endmodule

1. **仿真截图。**

****

1. **上板运行结果拍照。**

上板运行时，数码管累加跳动很慢，调小拨码开关代表的数值后即可加快跳动频率；如果跳动太快，也可以调大拨码开关代表的数值。



### 5 实验数据

表3.5调试实验数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 错误现象  （仿真现象或trace比对报错日志等） | 错误分析  （调试线索与依据） | 修改的代码 | 调试结果 |
| 271 | HCKTF9%%I3A)8VS]S5)JFLI | 出现Z形波，说明变量未被正确赋值，查找代码发现变量名写错 | 变量名从debug\_wb\_rf\_wen改为debug\_wb\_rf\_we | Z形波消失 |
| 273 | 4KG40TLN`XSS6FHBBW76G4Q | 查找wb\_rf\_wdata发现他被赋值final\_result，但其未被定义 | 添加wire [31:0] final\_result; | 成功赋值final\_result |
| 255 |  | 发现是引脚匹配有问题 | 修改成对应引脚 .alu\_src1 (alu\_src1 ), | 通过 |
| 220 | 1B84{8TGR[5W(@]G@0G$Z}C | 查找test文件发现pc直接到了bl的跳转地址。对于f\_we，bl是1，所以gr\_we应该没有~inst\_bl | assign gr\_we = ~inst\_st\_w & ~inst\_beq & ~inst\_bne & ~inst\_b;  //删除& ~inst\_bl | 通过 |
| 80  (alu.v) | IMG_256 | wb\_rf\_wdata值不对,查找test  文件，发现该指令是slli指令,rj应该是alu\_src1  逻c1辑左移指令 | assign sll\_result = alu\_src2 << alu\_sr[4:0]; 改为assign sll\_result = alu\_src1 << alu\_src2[4:0]; | 通过 |
| 74  (alu.v) | IMG_256 | wb\_rf\_wdata值不对。Or指令应该是操作数1和操作数2的或运算 | assign  or\_result = alu\_src1 |alu\_src2|alu\_result; 修改成：  assign  or\_result = alu\_src1 |alu\_src2; | 通过 |
| 83  (alu.v) | IMG_256 | wb\_rf\_wdata不对。查找test文件，发现该指令是srli指令,rj应为alu\_src1逻辑右移指令 | assign sr64\_result = {{32{op\_sra & alu\_src2[31]}}, alu\_src2[31:0]} >> alu\_src1[4:0];  改为：assign sr64\_result = {{32{op\_sra & alu\_src1[31]}}, alu\_src1[31:0]} >> alu\_src2[4:0]; | 通过 |
| 85  (alu.v) | IMG_256 | sr64\_result[30:0]出现了31位，错误较明显 | assign sr\_result = sr64\_result[30:0];改为assign sr\_result = sr64\_result[31:0]; | IMG_256 |

### 6 实验小结（包括遇到的问题及解决办法）

问题1：最后一个问题找了很久没找到

解决方法：询问其他同学，发现了sr64\_result[30:0]出现了31位的问题。通过这个问题，我深刻认识到寻找报错原因不应该只注重于操作码和指令的逻辑问题，还应该寻找较为基础的变量名称和数据位数问题。考虑问题应该更加全面，在遇到瓶颈时也应该寻求更多的方法。