单周期 CPU 仿真时序

SEU-09019204-曹邹颖

1. 用老师提供的汇编器 MinisysAv2.0.exe，在“工程”中选择 CPU 为 64KB，汇编测试文件 FINAL1.asm，汇编无误后在 output 文件夹中得到分别用于 rom 和 ram 的 COE 文件。

```
.DATA 0X0
    BUF0:   .word 0x55
    BUF1:   .word 0xAA

.TEXT 0X0000
start:
    lw   $v0,BUF0($zero)
    lw    $v1,BUF1($zero)
    add $at,$v0,$v1
    sw   $at,8($zero)
    subu $a0,$v1,$v0
    slt   $a0,$v0,$at
    and $at,$v1,$a3
    or    $a2,$v0,$at
    xor  $a3,$v0,$v1
    nor  $a2,$a1,$at
lop:
    beq $v1,$v0,lop
lop1:
    sub  $v0,$v0,$a1
    bne $a1,$v0,lop1
    beq $at,$at,lop2
    nop
lop2:
    jal   subp
    j     next
subp:
    jr    $ra
next:
    addi$v0,$zero,0x99
    ori   $v1,$zero,0x77
    sll   $v1,$v0,4
    srl   $v1,$v0,4
    srlv $v1,$v0,$at
    lui   $a2,0x9988
    sra   $a3,$a2,4
    addi$v0,$zero,0
    addi$v1,$zero,2
```

```
sub $at,$v0,$v1
lui  $a0,0xFFFF
ori  $a0,$a0,0xF000
lw   $v0,0xC70($a0)
sw   $v0,0xC60($a0)
j    start
```

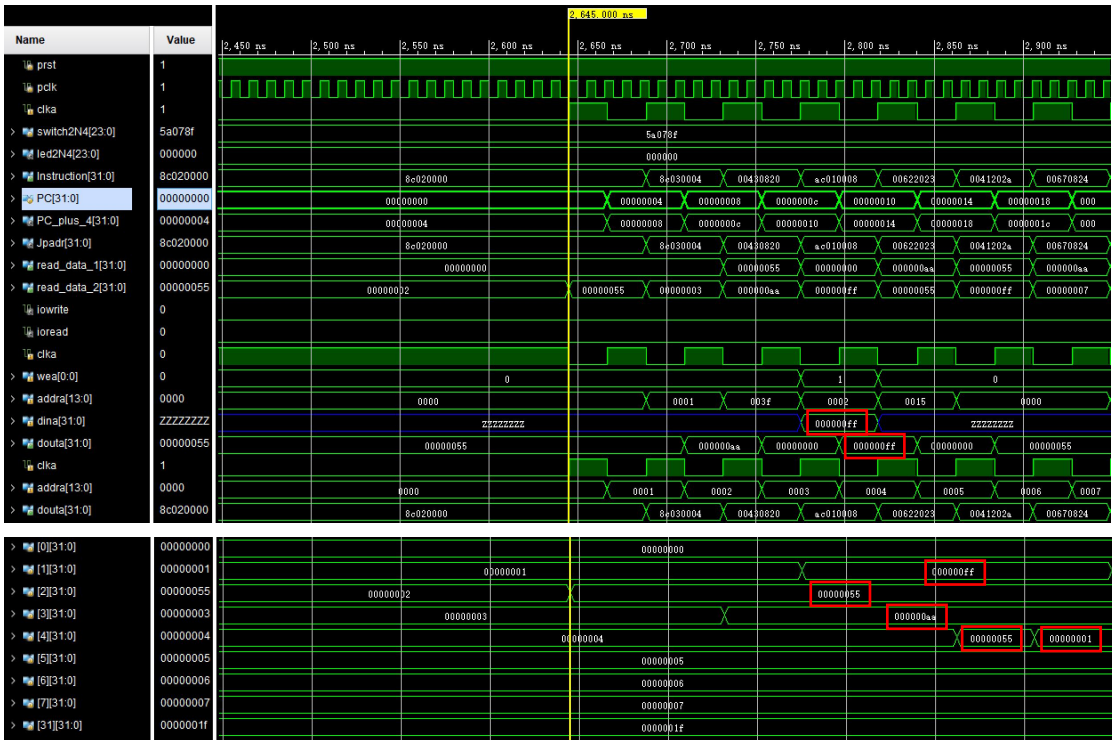
2. 设计的单周期 CPU 仿真波形图

①PC 从 00000000 开始，寄存器 i 初始值为 i。

```
.DATA 0X0
    BUF0:  .word 0x55
    BUF1:  .word 0xAA

.TEXT 0X0000
start:
    lw   $v0,BUF0($zero)
    lw    $v1,BUF1($zero)
    add $at,$v0,$v1
    sw   $at,8($zero)
    subu $a0,$v1,$v0
    slt   $a0,$v0,$at
```

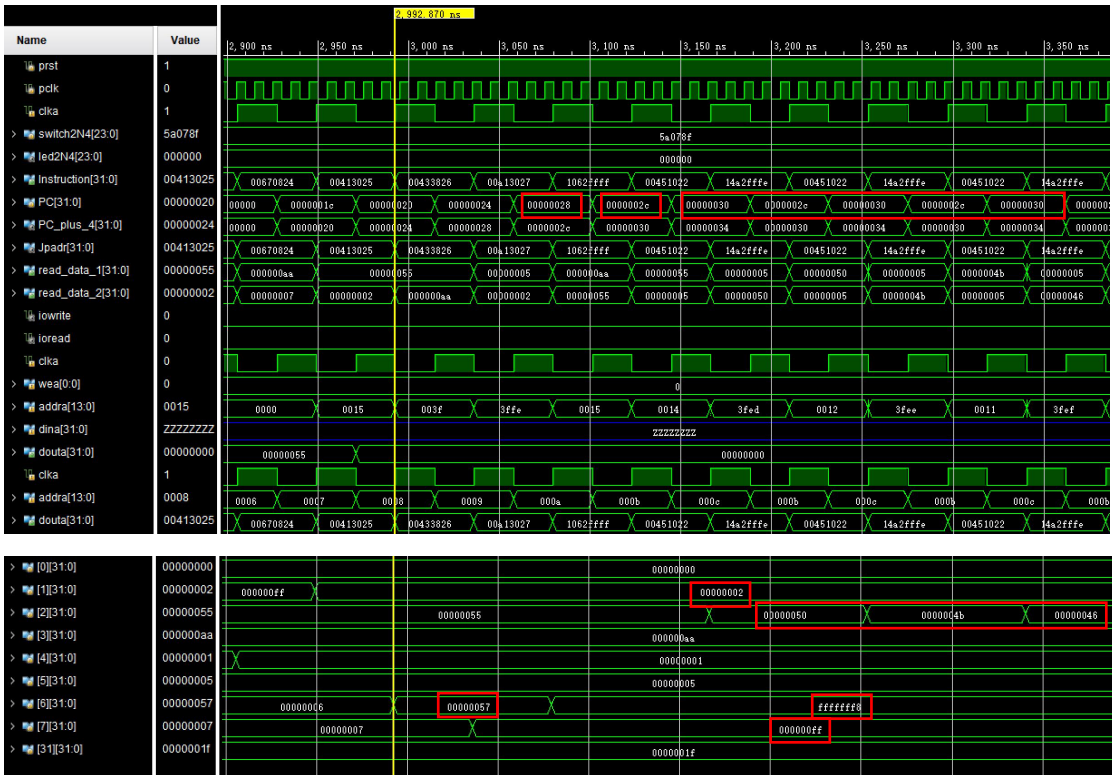
首先 lw $v0,BUF0($zero)完成寄存器$2 赋值 0x55，lw $v1,BUF1($zero)完成寄存器$3 赋值 0x55，add $at,$v0,$v1 完成寄存器$1 赋值($2)+($3)=0xFF，sw $at,8($zero)再将 0xFF 写入存储器中，然后 subu $a0,$v1,$v0 完成寄存器$4 赋值($3)-($2)=0x55。接着，slt $a0,$v0,$at 赋值寄存器$4 为 1(∵($2)<($1))。

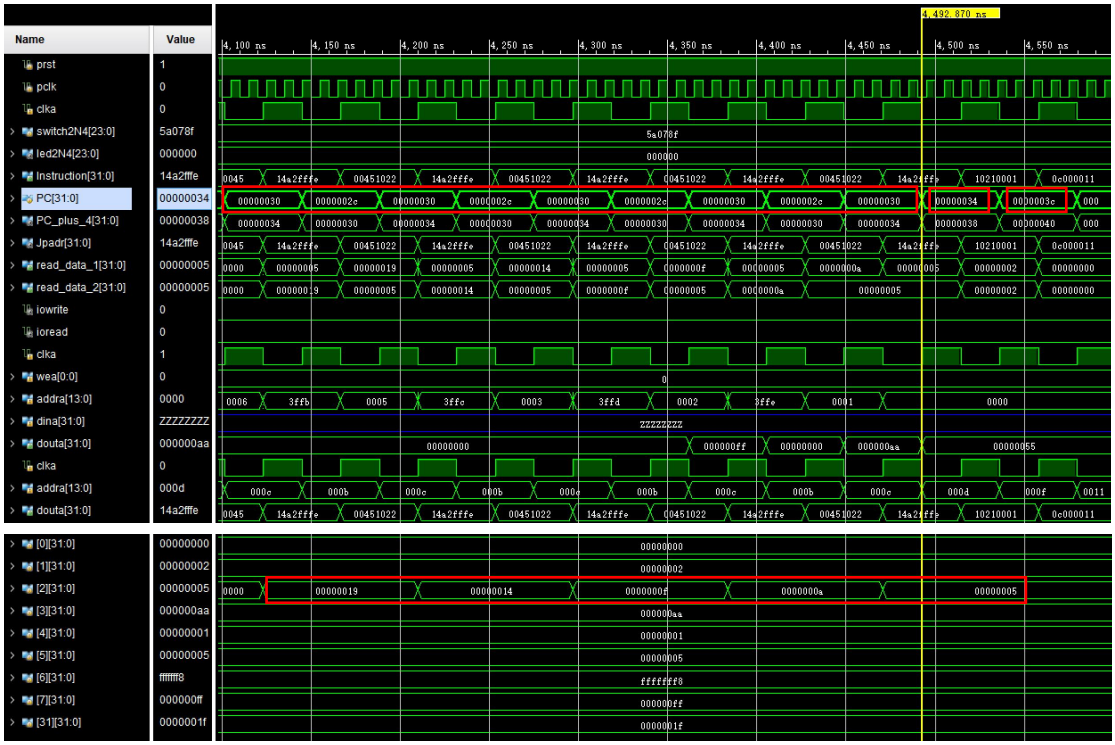


②and $at,$v1,$a3 赋值寄存器$1 为($3)&($7)=0x2， or $a2,$v0,$at 赋值寄存器$6 为($2)|($1)=0x57，xor $a3,$v0,$v1 赋值寄存器$7 为($2)^($3)=0xFF，nor $a2,$a1,$at 赋值寄存器$6 为~(($5)|($1))=0xFFFFFFF8。

```
      and $at,$v1,$a3
      or   $a2,$v0,$at
      xor  $a3,$v0,$v1
      nor  $a2,$a1,$at
lop:
      beq $v1,$v0,lop
lop1:
      sub $v0,$v0,$a1
      bne $a1,$v0,lop1
      beq $at,$at,lop2
      nop
lop2:
      jal   subp
```

beq $v1,$v0,lop 由于($3)≠($2)无需 PC 赋值为(PC)+4+((Sign-Extend)0xFFFF<<2)=(PC)+4-4=(PC)即死循环，所以 PC 正常+4。sub $v0,$v0,$a1 完成寄存器$2 赋值($2)-($5)=0x50，bne $a1,$v0,lop1 由于($5)≠($2)PC 跳转到(PC)+4+((Sign-Extend) 0xFFFE<<2)=(PC)+4-8=(PC)-4 即上一条指令 beq $v1,$v0,lop，由此循环执行，可见每次循环寄存器$2 的值减 5 直到其值为 0x5，再进行 bne $a1,$v0,lop1 指令时退出循环，PC 正常+4。beq $at,$at,lop2 由于($1)=($1)PC 跳转到(PC)+4+((Sign- Extend)0x1<<2)=(PC)+4+4=(PC)+8=0x3c 即下一条执行指令为 jal subp。



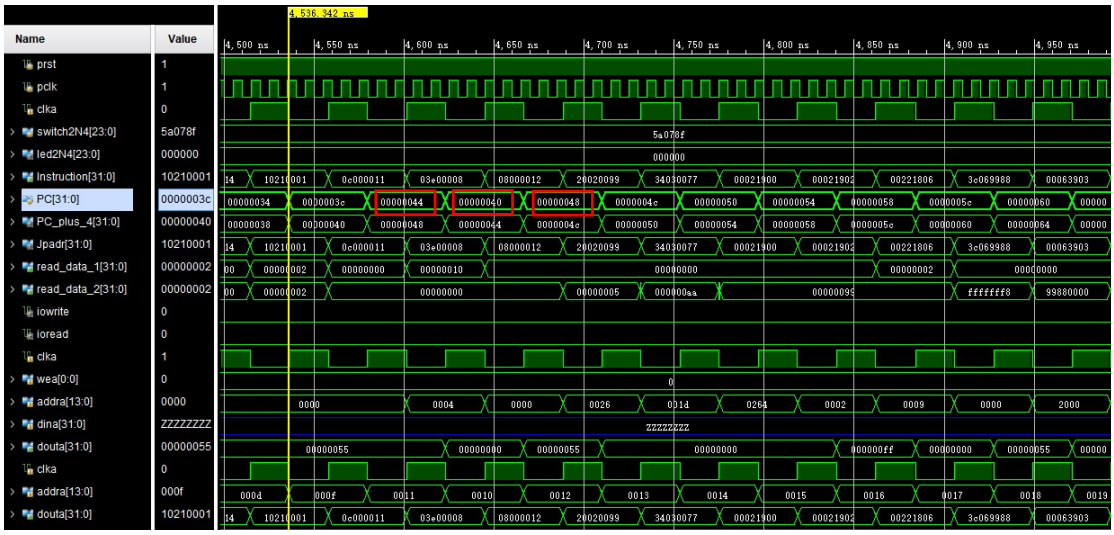


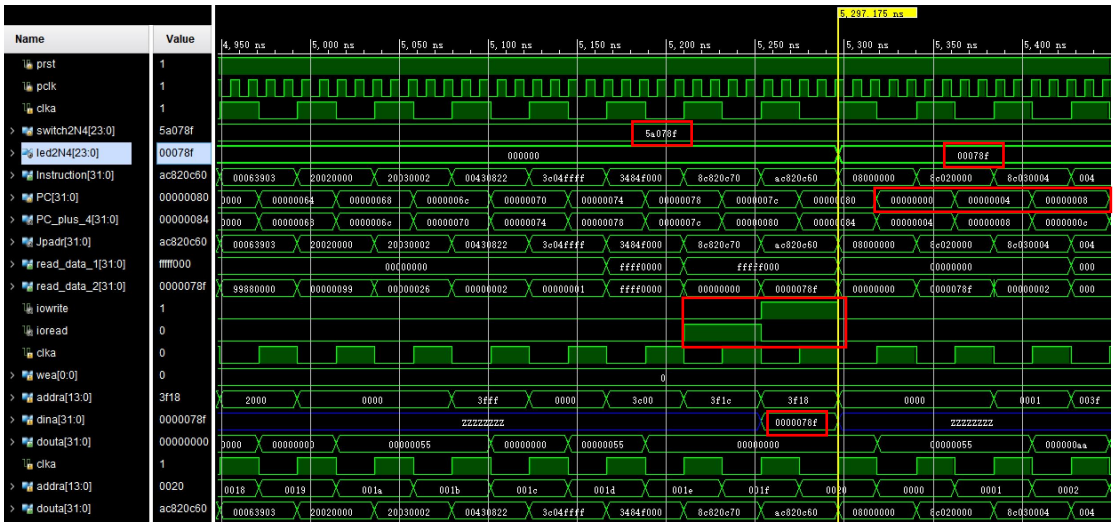
③jal subp 完成寄存器$31 赋值为(PC)+4=0x40 再右移两位为 0x10，同时 PC 跳转到((Zero-Extend) 0x11<<2)=0x44 即下一条执行指令为 jr $ra，jr $ra 完成 PC 跳转到($31)=0x40 即上一条指令，j next 完成 PC 跳转到((Zero-Extend)0x12<<2)=0x48 即下一条执行指令为 addi $v0,$zero,0x99。

```
lop2:
    jal   subp
    j     next
subp:
    jr    $ra
next:
    addi$v0,$zero,0x99
    ori   $v1,$zero,0x77
    sll   $v1,$v0,4
    srl   $v1,$v0,4
    srlv $v1,$v0,$at
    lui   $a2,0x9988
    sra   $a3,$a2,4
    addi$v0,$zero,0
    addi$v1,$zero,2
    sub  $at,$v0,$v1
    lui   $a0,0xFFFF
    ori   $a0,$a0,0xF000
    lw    $v0,0xC70($a0)
    sw    $v0,0xC60($a0)
    j     start
```

addi $v0,$zero,0x99 完成寄存器$2 赋值 0x99，ori $v1,$zero,0x77 完成寄存器$3 赋值 0x77，sll $v1,$v0,4 完成寄存器$3 赋值($2)<<4=0x990，srl $v1,$v0,4 完成寄存器$3 赋值($2)>>4=0x9，srlv $v1,$v0,$at 完成寄存器$3 赋值($2)<<($1)=($2)<<2=0x26，lui $a2,0x9988 完成寄存器$6 赋值 0x99880000，sra $a3,$a2,4 完成寄存器$7 赋值($6)>>4=0xF9988000。

addi$v0,$zero,0 完成寄存器$2 赋值 0x0，addi $v1,$zero,2 完成寄存器$3 赋值 0x2，sub $at,$v0,$v1 赋值寄存器$1 为($2)-($3)=0xFFFFFFFE，lui $a0,0xFFFF 赋值寄存器$4 为 0xFFFF0000，ori $a0,$a0,0xF000 完成寄存器$4 赋值 0xFFFFF000，lw $v0,0xC70($a0)与 sw $v0,0xC60($a0)做到读取拨码开关的值(5A078F)，并将其输出去，由于地址为 0xFFFFC60，写到 LED 模块的数据是拨码开关的低 16 位即 led2N4=0x78F，最后 PC 调转到 0。





3. minisys.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module minisys(prst,pclk,led2N4,switch2N4);
    input prst;                    //板上的 Reset 信号，低电平复位
    input pclk;                    //板上的 100MHz 时钟信号
    input[23:0] switch2N4;         //拨码开关输入
    output[23:0] led2N4;           //led 结果输出到 Nexys4 数码管

    wire clock;                    //clock: 分频后时钟供给系统
    wire iowrite,ioread;           //I/O 读写信号
    wire[31:0] write_data;         //写 RAM 或 IO 的数据
    wire[31:0] read_data;          //读 RAM 或 IO 的数据
    wire[15:0] ioread_data;        //读 IO 的数据
    wire[31:0] pc_plus_4;          //PC+4
    wire[31:0] read_data_1;        //译码单元读出的数据 1
    wire[31:0] read_data_2;        //译码单元读出的数据 2
    wire[31:0] sign_extend;        //符号扩展
    wire[31:0] add_result;
    wire[31:0] alu_result;
    wire[31:0] mread_data;         //RAM 中读取的数据
    wire[31:0] address;
    wire alusrc;
    wire branch;
    wire nbranch,jmp,jal,jrn,i_format;
    wire regdst;
    wire regwrite;
    wire zero;
    wire memwrite;
    wire memread;
    wire memoriotoreg;
    wire memreg;
    wire sftmd;
    wire[1:0] aluop;
    wire[31:0] instruction;
    wire[31:0] opcplus4;
    wire ledctrl,switchctrl;
    wire[15:0] ioread_data_switch;
```

```
cpuclk cpuclk(
        .clk_in1(pclk),             // 100MHz
        .clk_out1(clock)          // cpuclock,23MHz
);

Ifetc32 ifetch(
        .Instruction(instruction),
        .PC_plus_4_out(pc_plus_4),
        .Add_result(add_result),
        .Read_data_1(read_data_1),
        .Branch(branch),
        .nBranch(nbranch),
        .Jmp(jmp),
        .Jal(jal),
        .Jrn(jrn),
        .Zero(zero),
        .clock(clock),
        .reset(!prst),
        .opcplus4(opcplus4)
);

Idecode32 idecode(
        .read_data_1(read_data_1),
        .read_data_2(read_data_2),
        .Instruction(instruction),
        .read_data(read_data),        //  从 DATA RAM or I/O port 取出的数据
        .ALU_result(alu_result),
        .Jal(jal),
        .RegWrite(regwrite),
        .MemIOtoReg(memoriotoreg),
        .RegDst(regdst),
        .Sign_extend(sign_extend),
        .clock(clock),
        .reset(!prst),
        .opcplus4(opcplus4)
        //.read_register_1_address(instruction[25:21])//rs
);

control32 control(
        .Opcode(instruction[31:26]),
        .Function_opcode(instruction[5:0]),
        .Alu_resultHigh(alu_result[31:10]),
        .Jrn(jrn),
        .RegDST(regdst),
```

```
        .ALUSrc(alusrc),
        .MemIOtoReg(memoriotoreg),
        .RegWrite(regwrite),
        .MemRead(memread),
        .MemWrite(memwrite),
        .IORead(ioread),
        .IOWrite(iowrite),
        .Branch(branch),
        .nBranch(nbranch),
        .Jmp(jmp),
        .Jal(jal),
        .I_format(i_format),
        .Sftmd(sftmd),
        .ALUOp(aluop)
    );

    Executs32 execute(
        .Read_data_1(read_data_1),
        .Read_data_2(read_data_2),
        .Sign_extend(sign_extend),
        .Function_opcode(instruction[5:0]), // func
        .Exe_opcode(instruction[31:26]), //op code
        .ALUOp(aluop),
        .Shamt(instruction[10:6]),
        .ALUSrc(alusrc),
        .I_format(i_format),
        .Zero(zero),
        .Sftmd(sftmd),
        .ALU_Result(alu_result),
        .Add_Result(add_result),
        .PC_plus_4(pc_plus_4)
     );

    memorio memio( // IO MEM 统一编址
        .caddress(alu_result), // from alu_result in executs32
        .address(address), // output, address to DMEM
        .memread(memread), // read memory, from control32
        .memwrite(memwrite), // write memory, from control32
        .ioread(ioread), // read IO, from control32
        .iowrite(iowrite), // write IO, from control32
        .mread_data(mread_data), // data from memory
        .ioread_data(ioread_data), // data from io
        .wdata(read_data_2),// the data from idecode32,that want to write memio
        .rdata(read  data), // ouput,mread  data 与 ioread  data 选其一
```

```
        .write_data(write_data),    // output,data to memory or I/O
        .LEDCtrl(ledctrl),
        .SwitchCtrl(switchctrl)
    );

    dmemory32 memory(                    //  数据 RAM
      .read_data(mread_data),            // RAM 读出
      .address(address),
      .write_data(write_data),
      .Memwrite(memwrite),
      .clock(clock)
    );

    ioread multiioread(
        .reset(!prst),///
        .clk(clock),///
        .ior(ioread),
        .switchctrl(switchctrl),
        .ioread_data(ioread_data),// output
        .ioread_data_switch(ioread_data_switch)// input
    );

    leds led16(
        .led_clk(clock),
        .ledrst(!prst),////
        .ledwrite(ledctrl),
        .ledcs(ledctrl),
        .ledaddr(address[1:0]),
        .ledwdata(write_data[15:0]),
        .ledout(led2N4)// output
      );

    switchs switch16(
        .switclk(clock),
        .switrst(!prst),///
        .switchread(switchctrl),
        .switchaddr(address[1:0]),
        .switchcs(switchctrl),
        .switchrdata(ioread_data_switch),// output
        .switch_i(switch2N4)// input
      );

endmodule
```