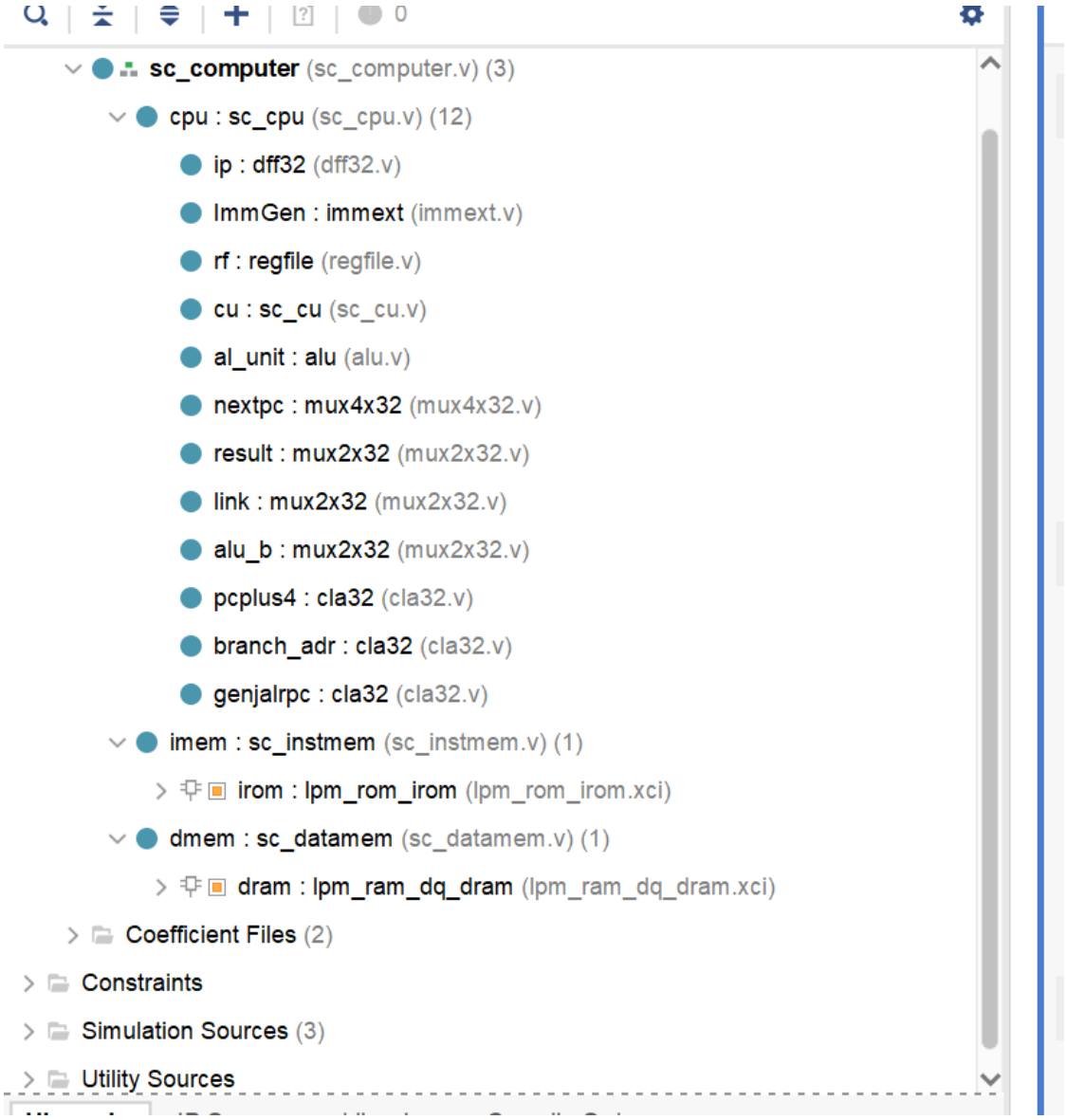
Lab 3 实验报告

521030910087 吴舒文

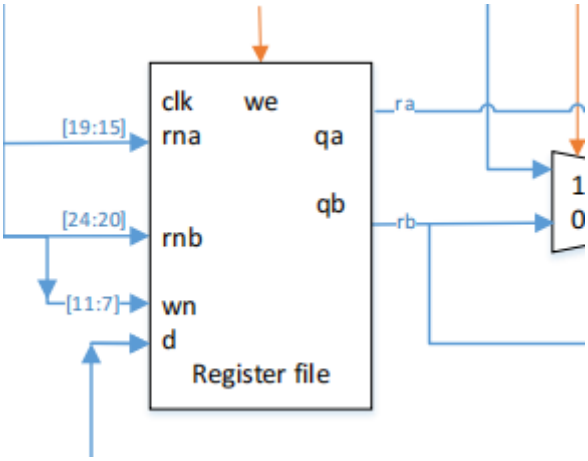
1. 实验目的
    1. 掌握不同类型指令在数据通路中的执行路径。
    2. 掌握Vivado 仿真方式。
2. 设计思路
    1. 从前序实验中导入已完成的各个模块本次实验中所需完成的模块，并完成指令rom 和数据 ram 的例化



    2. 实现sc_cpu.v

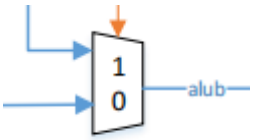
```
//register file,    mux2x32 , regfile
   regfile rf(
   .clk(clock),
   .clrn(resetn),
   .rna(inst[19:15]),
   .rnb(inst[24:20]),
   .qa(ra),
   .qb(data),
   .wn(inst[11:7]),
   .we(wreg),
   .d(regf_din)
);
```

1. 寄存器



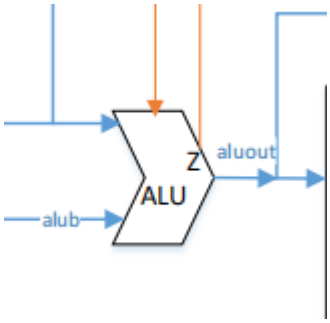
2. 用aluimm决定alub

```
mux2x32 alu_b(
   .s(aluimm),
   .a1(immediate),
   .a0(data),
   .y(alub)
);
```

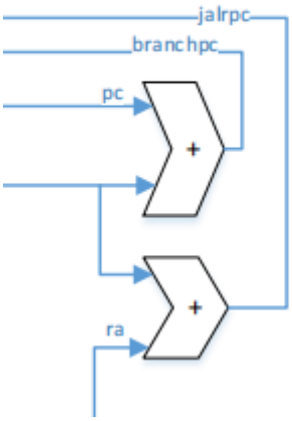


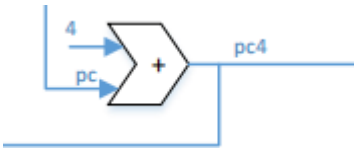
3. alu

```
alu al_unit(
      .a(ra),
      .b(alub),
      .aluc(aluc),
      .s(aluout),
      .z(zero)
    );
```



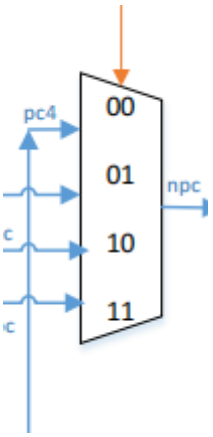
4. 三个加法器 分别计算分支/跳转/顺序情况下的下一条指令地址、





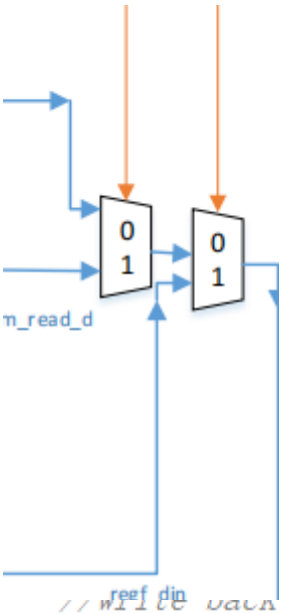

```
cla32 pcplus4(
  .pc(pc),
  .x1(32'h4),
  .p4(p4)
);

cla32 branch_adr(
  .pc(pc),
  .x1(immediate),
  .p4(branchpc)
);

cla32 genjalrpc(
  .pc(ra),
  .x1(immediate),
  .p4(jalrpc)
);
```

5. 四路选择器 根据 pcsource 给出下一条指令地址




```
mux4x32 nextpc(
  .s(pcsource),
  .a0(p4),
  .a1(branchpc),
  .a2(jalrpc),
  .a3(branchpc),
  .y(npc)
);

wire [31:0]tmp;
```

6. 两个二路选择器 根据 m2reg 和 pcsource[1]决定写回寄存器的内容



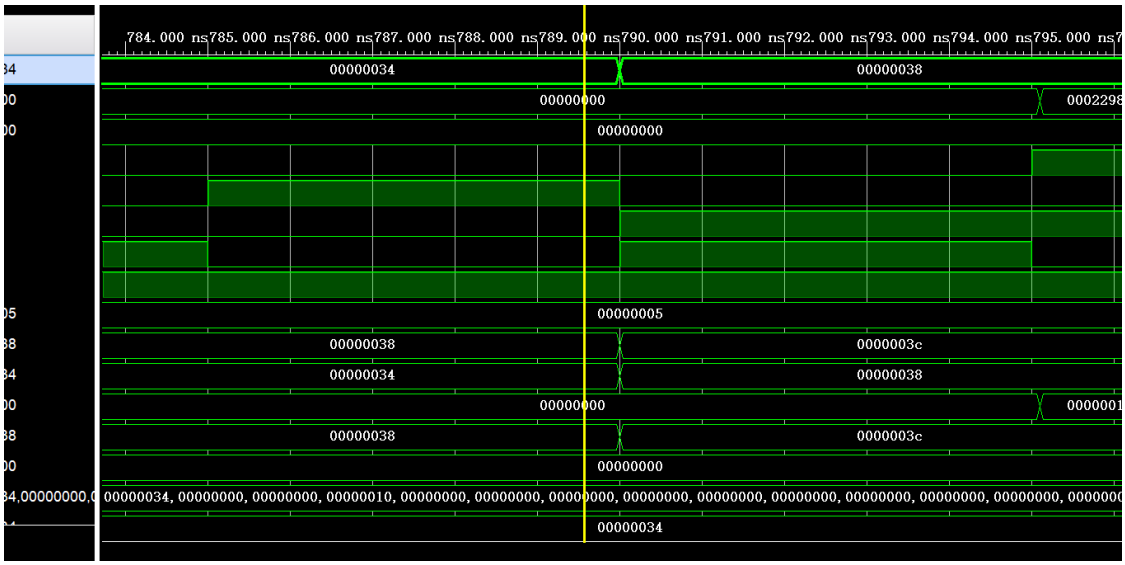

```
//write back register file,    mux2x32
mux2x32 result(
  .s(m2reg),
  .a1(mem),
  .a0(aluout),
  .y(tmp)
);

mux2x32 link(
  .s(pcsource[1]),
  .a0(tmp),
  .a1(p4),
  .y(regf_din)
);
```

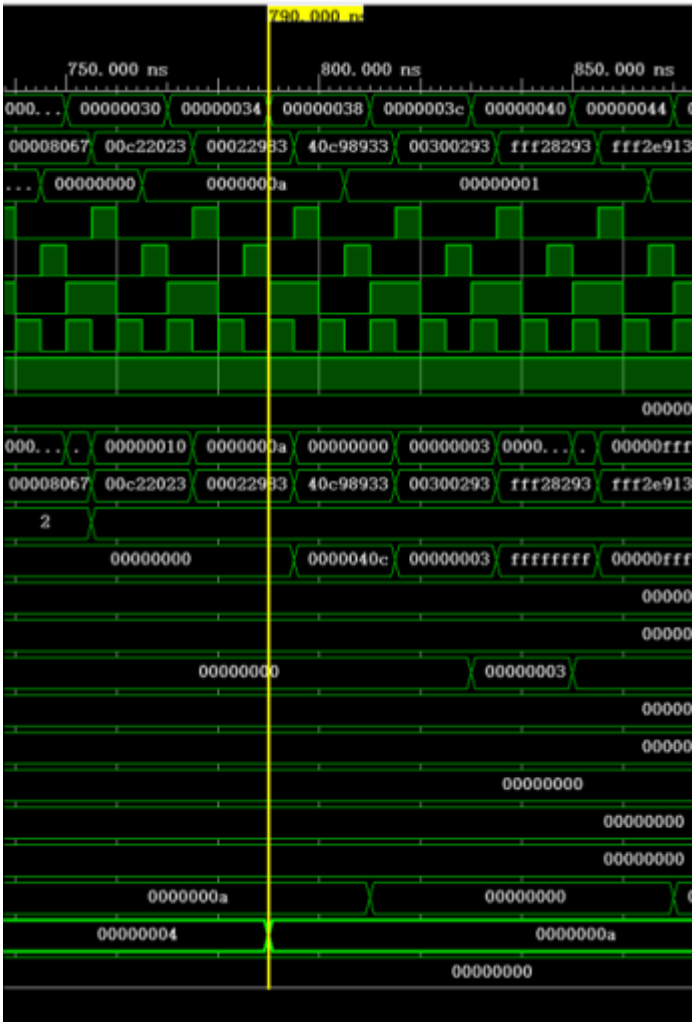
3. 进行仿真

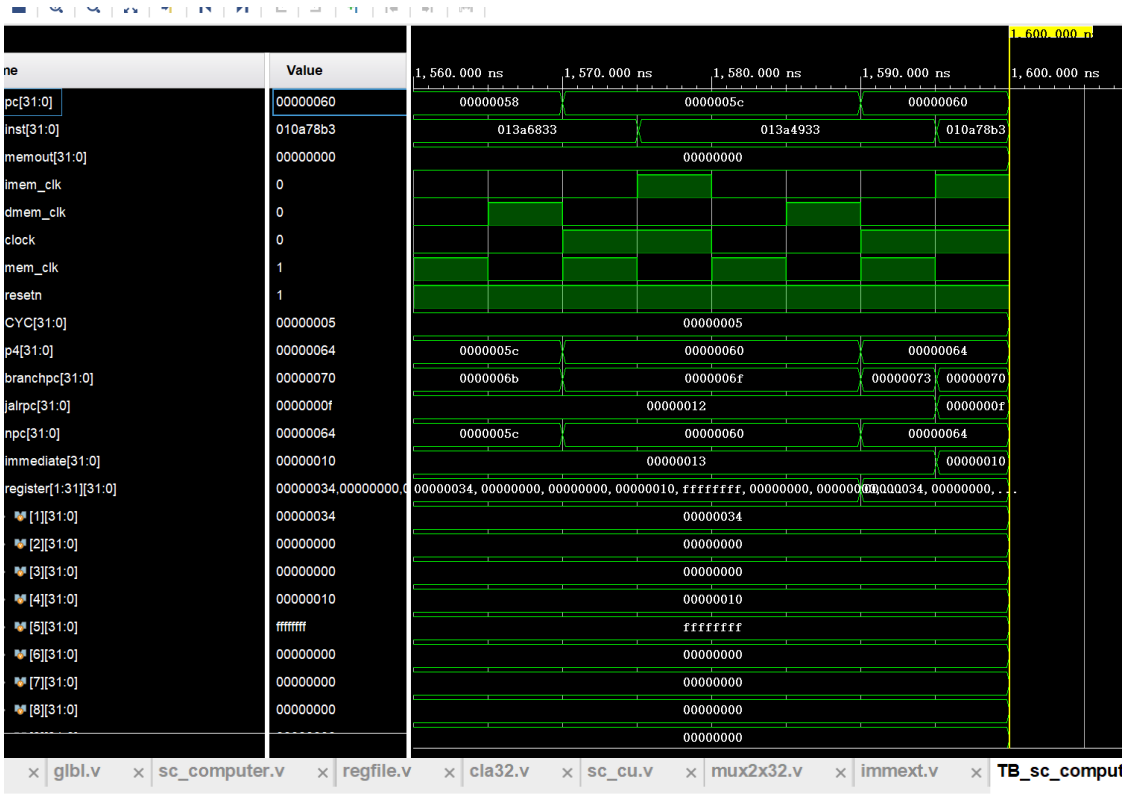
   将仿真时长调至1600ns，得到如下结果

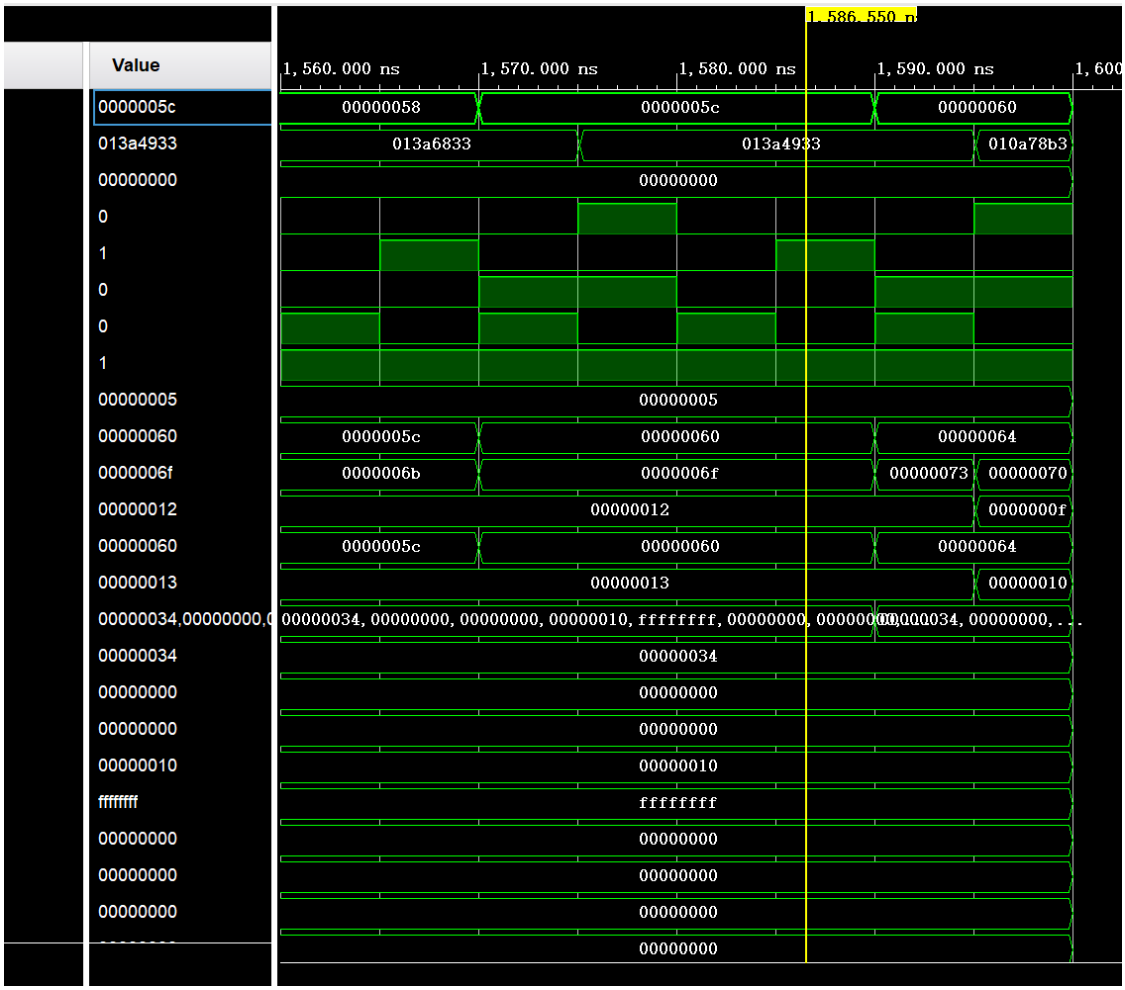
   由此得出1515ns指令执行结束，最后的指令地址为0000007c；最后一条指令为0000006f；此时x18寄存器的值为0001ffff，与指导书结果一致。

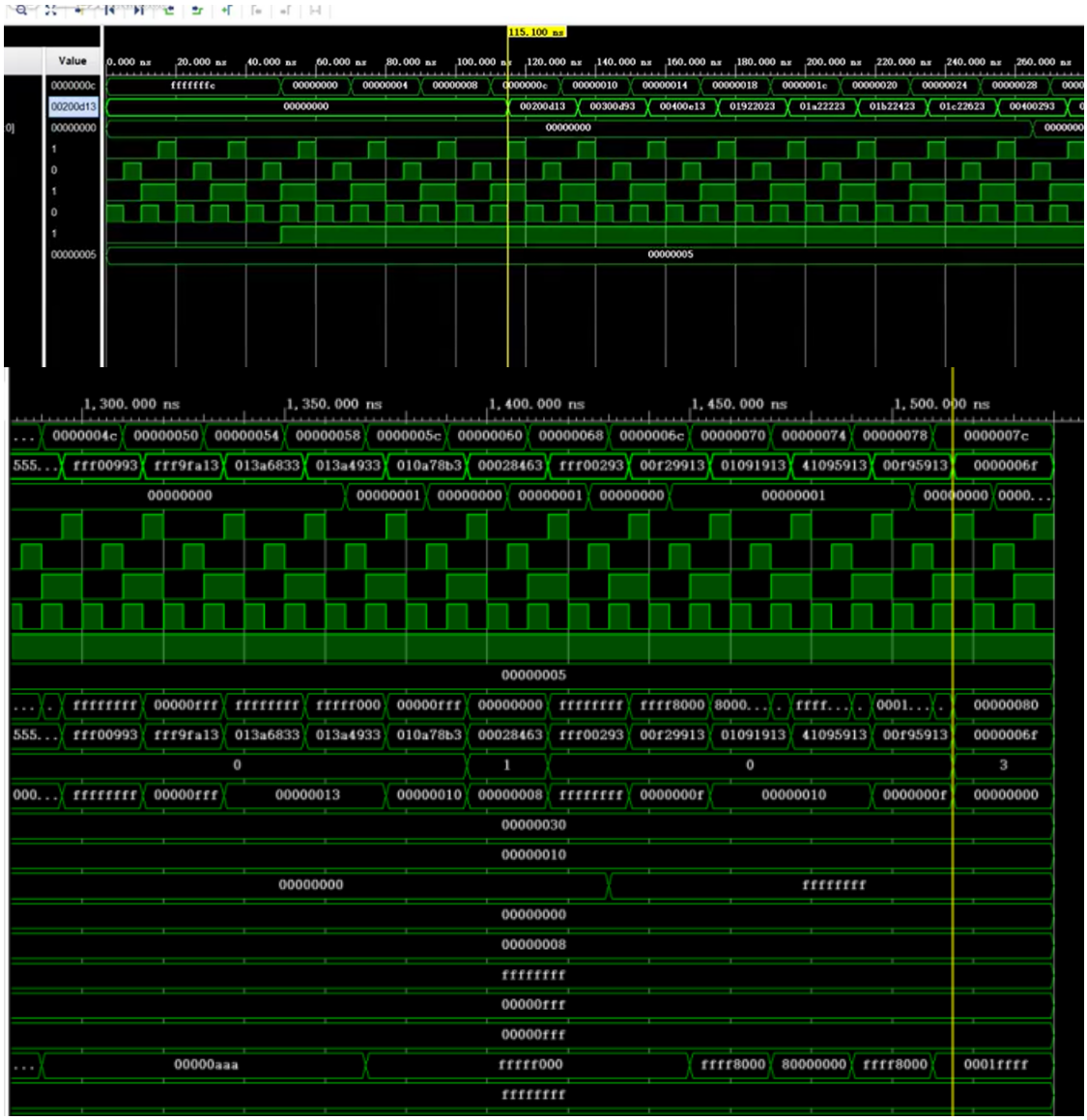
   790ns后，寄存器x19的值为0000000a，说明sw,lw执行正确


784. 000 ns
785. 000 ns
786. 000 ns
787. 000 ns
788. 000 ns
789. 000 ns
790. 000 ns
791. 000 ns
792. 000 ns
793. 000 ns
794. 000 ns
795. 000 ns
00000034
00000038
00000000
00000000
00000005
00000038
0000003c
00000034
00000038
00000000
00000038
0000003c
00000000
00000034, 00000000, 00000000, 00000010, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000
00000034


750. 000 ns
800. 000 ns
850. 000 ns
00000030
00000034
00000038
0000003c
00000040
00000044
00008067
00c22023
00022983
40c98933
00300293
fff28293
00000000
0000000a
00000001
00000010
00000000
00000003
00000000
00000000
0000040c
00000003
ffffffff
00000003
0000000a
00000000
00000004
0000000a
00000000


Value
pc[31:0] 00000060
inst[31:0] 010a78b3
memout[31:0] 00000000
imem_clk 0
dmem_clk 0
clock 0
mem_clk 1
resetn 1
CYC[31:0] 00000005
p4[31:0] 00000064
branchpc[31:0] 00000070
jalrpc[31:0] 0000000f
npc[31:0] 00000064
immediate[31:0] 00000010
register[1:31][31:0]
[1][31:0] 00000034
[2][31:0] 00000000
[3][31:0] 00000000
[4][31:0] 00000010
[5][31:0] ffffffff
[6][31:0] 00000000
[7][31:0] 00000000
[8][31:0] 00000000
1,560.000 ns
1,570.000 ns
1,580.000 ns
1,590.000 ns
1,600.000 ns
glbl.v
sc_computer.v
regfile.v
cla32.v
sc_cu.v
mux2x32.v
immext.v
TB_sc_comput


Value
0000005c
013a4933
00000000
0
1
0
0
1
00000005
00000060
0000006f
00000012
00000060
00000013
00000034,00000000,
00000034
00000000
00000000
00000010
ffffffff
00000000
00000000
00000000
1,560.000 ns
1,570.000 ns
1,580.000 ns
1,590.000 ns
1,600



4. 拓展思考：对比Ripes软件生成的框架图，本次完成的数据通路其实使用了许多不必要的 FPGA 资源，考虑尽量减少使用硬件资源，用于产生branchpc 和jalrpc 地址的加法器是可以省掉的，同样使用ALU来计算目标地址，这样也能省掉四选一多路选择器。最终我们仅使用二选一多路选择器即可完成完整的数据通路。

   Ripes使用的是二路选择器来完成数据通路，用于用于产生branchpc 和jalrpc 地址的加法器也省掉了。