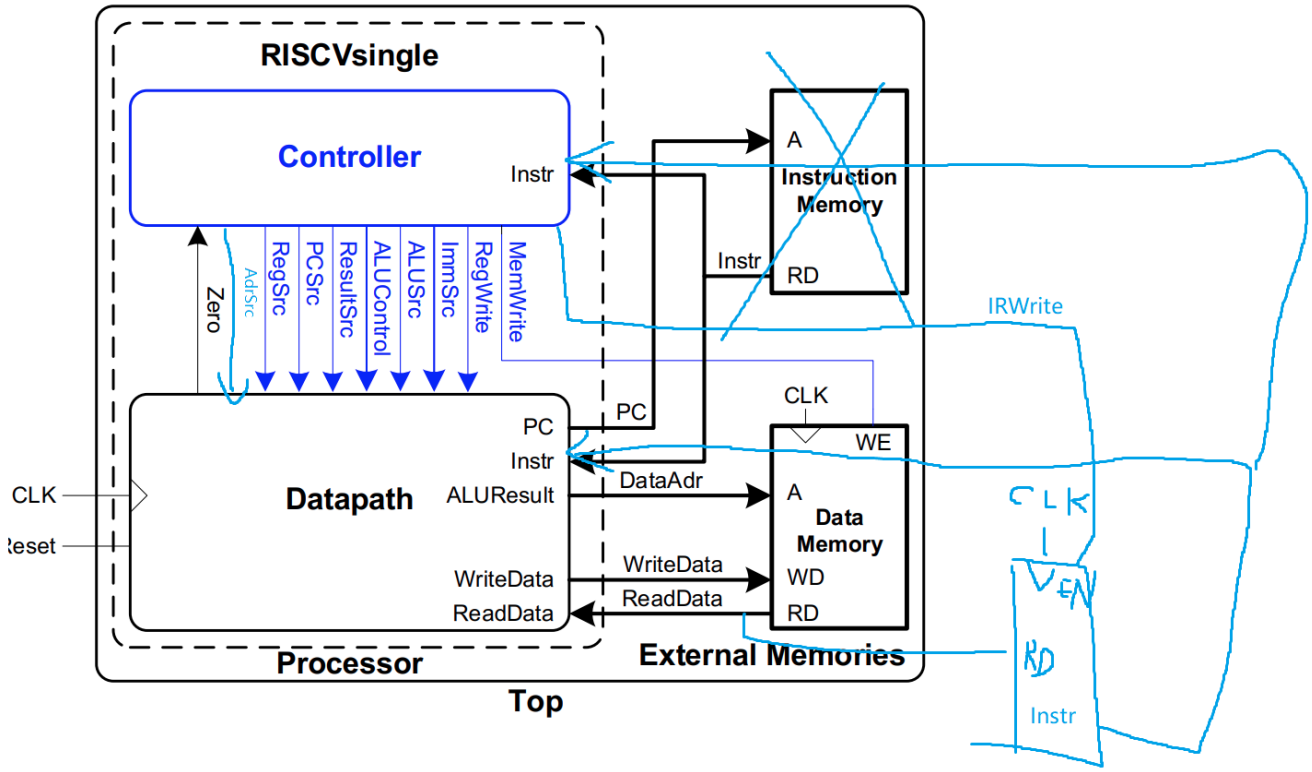
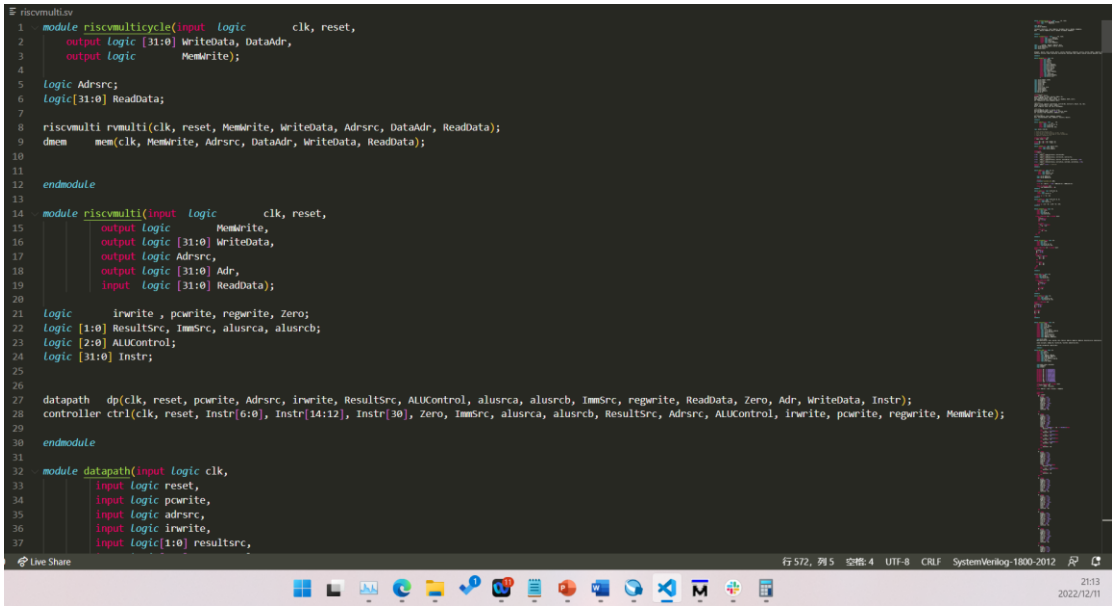
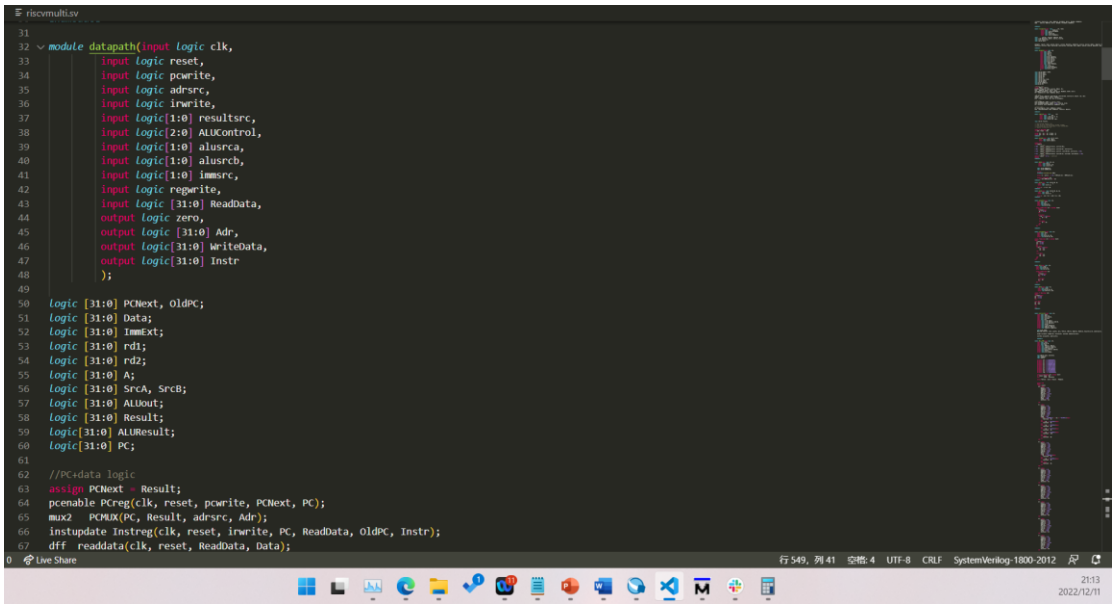
1. I spent 6.5h.

2. Diagram

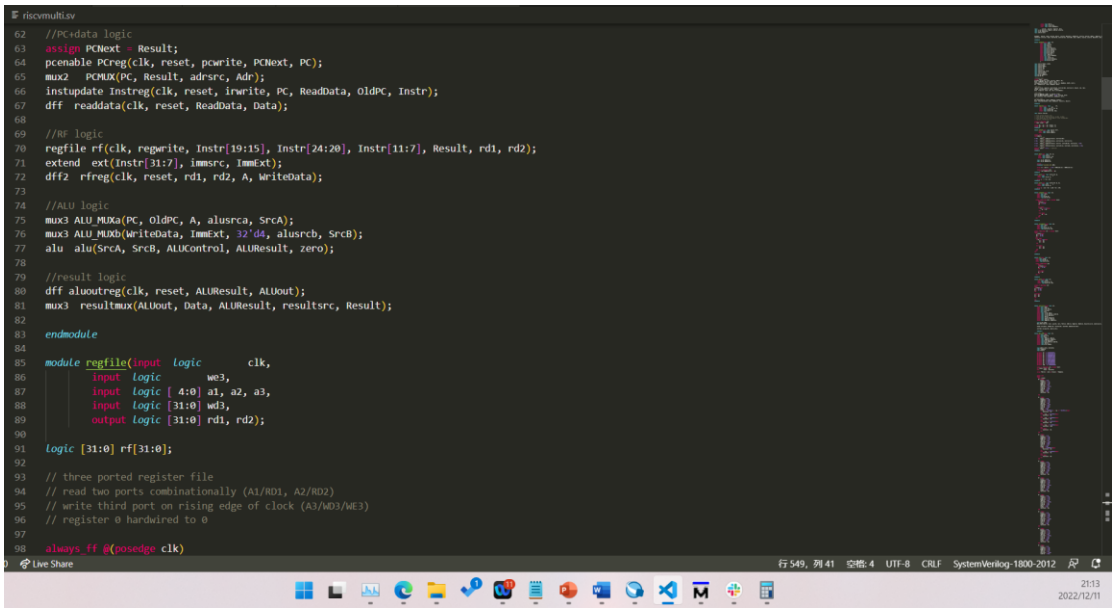




```
riscvmulti.sv
module riscvmulticycle(input  logic        clk, reset,
    output logic [31:0] WriteData, DataAdr,
    output logic        MemWrite);

logic Adrsrc;
logic[31:0] ReadData;

riscvmulti rvmulti(clk, reset, MemWrite, WriteData, Adrsrc, DataAdr, ReadData);
dmem     mem(clk, MemWrite, Adrsrc, DataAdr, WriteData, ReadData);


endmodule

module riscvmulti(input  logic        clk, reset,
          output logic        MemWrite,
          output logic [31:0] WriteData,
          output logic Adrsrc,
          output logic [31:0] Adr,
          input  logic [31:0] ReadData);

logic       irwrite , pcwrite, regwrite, zero;
logic [1:0] ResultSrc, ImmSrc, alusrca, alusrcb;
logic [2:0] ALUControl;
logic [31:0] Instr;


datapath   dp(clk, reset, pcwrite, Adrsrc, irwrite, ResultSrc, ALUControl, alusrca, alusrcb, ImmSrc, regwrite, ReadData, zero, Adr, WriteData, Instr);
controller ctrl(clk, reset, Instr[6:0], Instr[14:12], Instr[30], zero, ImmSrc, alusrca, alusrcb, ResultSrc, Adrsrc, ALUControl, irwrite, pcwrite, regwrite, MemWrite);

endmodule

module datapath(input logic clk,
          input logic reset,
          input logic pcwrite,
          input logic adrsrc,
          input logic irwrite,
          input logic[1:0] resultsrc,
```



```
riscvmulti.sv
module datapath(input logic clk,
          input logic reset,
          input logic pcwrite,
          input logic adrsrc,
          input logic irwrite,
          input logic[1:0] resultsrc,
          input logic[2:0] ALUControl,
          input logic[1:0] alusrca,
          input logic[1:0] alusrcb,
          input logic[1:0] immsrc,
          input logic regwrite,
          input logic [31:0] ReadData,
          output logic zero,
          output logic [31:0] Adr,
          output logic[31:0] WriteData,
          output logic[31:0] Instr
          );

  logic [31:0] PCNext, OldPC;
  logic [31:0] Data;
  logic [31:0] ImmExt;
  logic [31:0] rd1;
  logic [31:0] rd2;
  logic [31:0] A;
  logic [31:0] SrcA, SrcB;
  logic [31:0] ALUout;
  logic [31:0] Result;
  logic[31:0] ALUResult;
  logic[31:0] PC;

  //PC+data logic
  assign PCNext = Result;
  pcenable PCreg(clk, reset, pcwrite, PCNext, PC);
  mux2   PCMUX(PC, Result, adrsrc, Adr);
  instupdate Instreg(clk, reset, irwrite, PC, ReadData, OldPC, Instr);
  dff  readdata(clk, reset, ReadData, Data);
```



```
riscvmulti.sv
  //PC+data logic
  assign PCNext = Result;
  pcenable PCreg(clk, reset, pcwrite, PCNext, PC);
  mux2   PCMUX(PC, Result, adrsrc, Adr);
  instupdate Instreg(clk, reset, irwrite, PC, ReadData, OldPC, Instr);
  dff  readdata(clk, reset, ReadData, Data);

  //RF logic
  regfile rf(clk, regwrite, Instr[19:15], Instr[24:20], Instr[11:7], Result, rd1, rd2);
  extend  ext(Instr[31:7], immsrc, ImmExt);
  dff2  rfreg(clk, reset, rd1, rd2, A, WriteData);

  //ALU logic
  mux3 ALU_MUXa(PC, OldPC, A, alusrca, SrcA);
  mux3 ALU_MUXb(WriteData, ImmExt, 32'd4, alusrcb, SrcB);
  alu  alu(SrcA, SrcB, ALUControl, ALUResult, zero);

  //result logic
  dff aluoutreg(clk, reset, ALUResult, ALUout);
  mux3  resultmux(ALUout, Data, ALUResult, resultsrc, Result);

  endmodule

  module regfile(input  logic        clk,
          input  logic        we3,
          input  logic [ 4:0] a1, a2, a3,
          input  logic [31:0] wd3,
          output logic [31:0] rd1, rd2);

  logic [31:0] rf[31:0];

  // three ported register file
  // read two ports combinationally (A1/RD1, A2/RD2)
  // write third port on rising edge of clock (A3/WD3/WE3)
  // register 0 hardwired to 0

  always_ff @(posedge clk)
```

riscvmulti.sv

```
98   always_ff @(posedge clk)
99   if (we3) rf[a3] <= wd3;
100
101  assign rd1 = (a1 != 0) ? rf[a1] : 0;
102  assign rd2 = (a2 != 0) ? rf[a2] : 0;
103  endmodule
104
105  module extend(input  logic [31:7] instr,
106                input  logic [1:0]  immsrc,
107                output logic [31:0] immext);
108
109  always_comb
110  case(immsrc)
111              // I-type
112  2'b00:   immext = {{20{instr[31]}}, instr[31:20]};
113             // S-type (stores)
114  2'b01:   immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};
115            // B-type (branches)
116  2'b10:   immext = {{19{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
117           // J-type (jal)
118  2'b11:   immext = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
119           //  U-type
120  default: immext = 32'bx; // undefined
121  endcase
122  endmodule
123
124
125  module dmem(input  logic clk, we,
126              input  logic adrsrc,
127              input  logic [31:0] a, wd,
128              output logic [31:0] rd);
129
130    logic [31:0] RAM[63:0];
131    logic [31:0] dRAM[63:0];
132
133    initial
134    $readmemh("riscvtest.txt".RAM);
```

Live Share 行 572，列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012

21:13
2022/12/11

riscvmulti.sv

```
132
133    initial
134    $readmemh("riscvtest.txt",RAM);
135
136    assign rd = (adrsrc == 1'b0) ? RAM[a[31:2]] : dRAM[a[31:2]];
137
138    always_ff @(posedge clk)
139      if (we) dRAM[a[31:2]] <= wd;
140  endmodule
141
142  module mux2(input  logic [31:0] d0, d1,
143              input  logic        s,
144              output logic [31:0] y);
145
146    assign y = s ? d1 : d0;
147  endmodule
148
149  module mux3(input  logic [31:0] d0, d1, d2,
150              input  logic [1:0]  s,
151              output logic [31:0] y);
152
153    assign y = s[1] ? d2 : (s[0] ? d1 : d0);
154  endmodule
155
156
157  module pcenable(input logic clk,
158      input logic reset,
159      input logic en,
160      input logic[31:0] d1,
161      output logic[31:0] d2);
162
163   always_ff@(posedge clk or posedge reset)
164      begin
165       if(reset)begin
166        d2 <= 32'b0;
167       end
168
```

Live Share 行 572，列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012

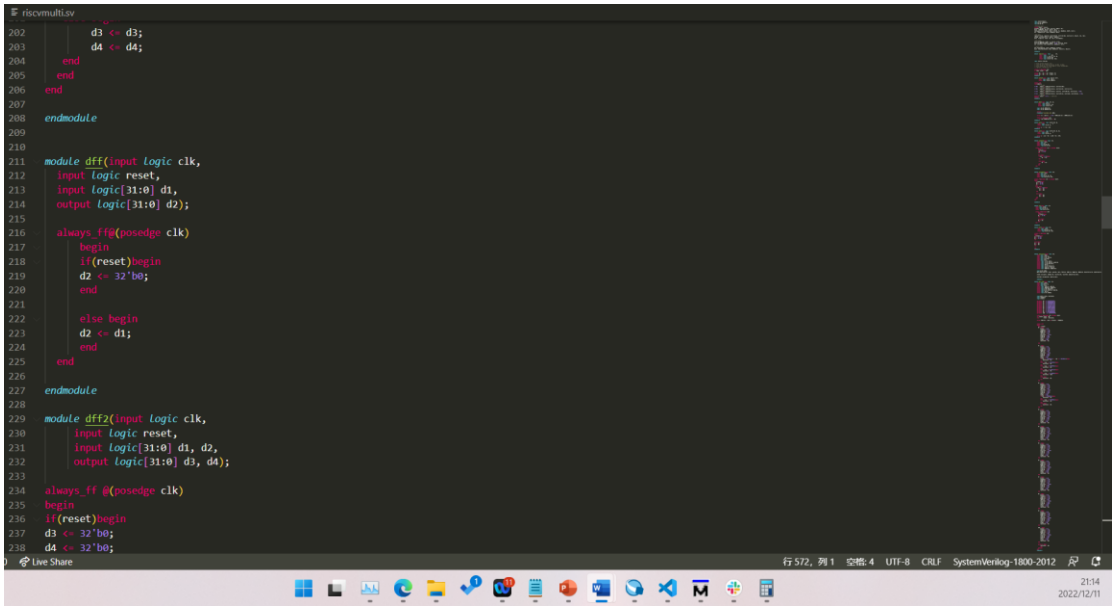
21:14
2022/12/11

riscvmulti.sv

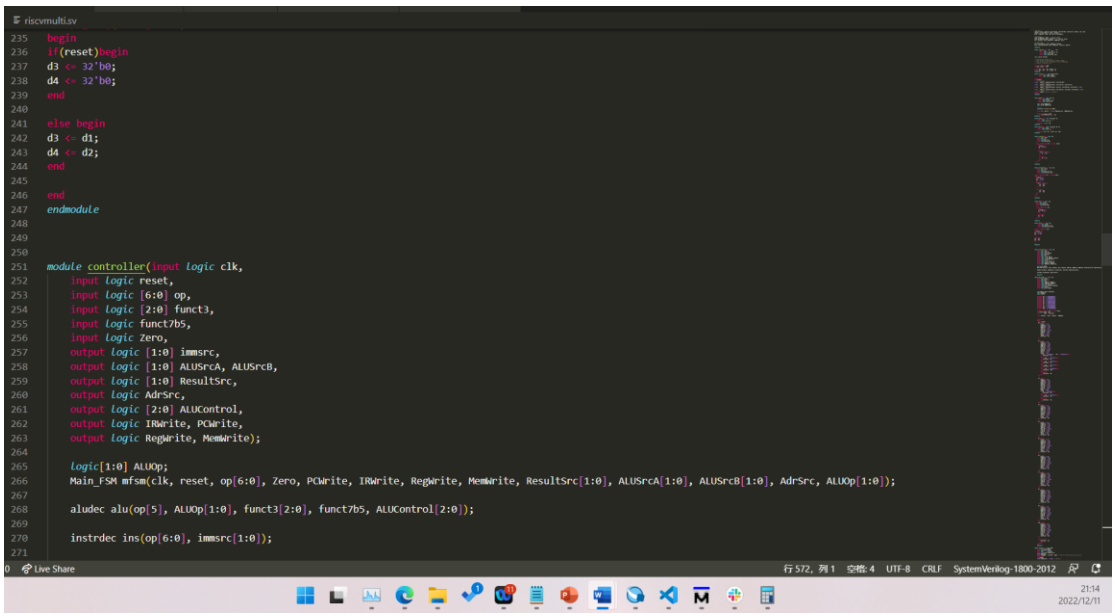
```
169       else begin
170        if(en == 1'b1)begin
171            d2 <= d1;
172        end
173
174        else begin
175            d2 <= d2;
176        end
177       end
178    end
179
180    endmodule
181
182
183    module instupdate(input logic clk,
184       input logic reset,
185       input logic en,
186       input logic[31:0] d1, d2,
187       output logic[31:0] d3, d4);
188
189   always_ff@(posedge clk or posedge reset)
190   begin
191     if(reset)begin
192      d3 <= 32'b0;
193      d4 <= 32'b0;
194     end
195     else begin
196      if(en == 1'b1)begin
197          d3 <= d1;
198          d4 <= d2;
199      end
200
201      else begin
202          d3 <= d3;
203          d4 <= d4;
204      end
205    end
```

Live Share 行 572，列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012

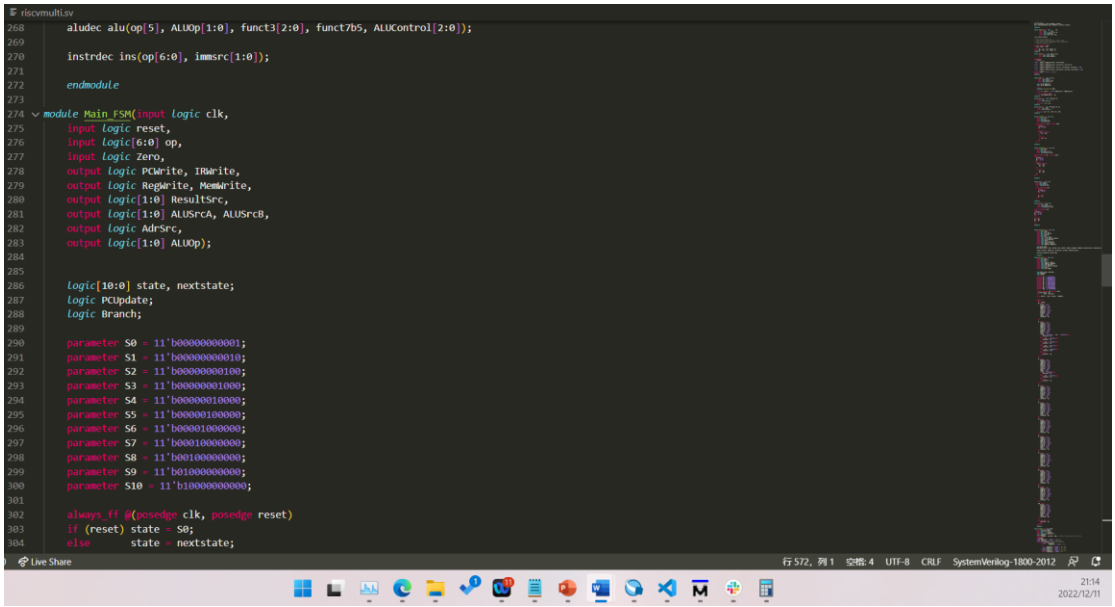
21:14
2022/12/11



```
riscvmulti.sv
202             d3 <= d3;
203             d4 <= d4;
204         end
205       end
206     end
207
208     endmodule
209
210
211     module dff(input logic clk,
212       input logic reset,
213       input logic[31:0] d1,
214       output logic[31:0] d2);
215
216       always_ff@(posedge clk)
217           begin
218           if(reset)begin
219           d2 <= 32'b0;
220           end
221
222           else begin
223           d2 <= d1;
224           end
225       end
226
227     endmodule
228
229     module dff2(input logic clk,
230           input logic reset,
231           input logic[31:0] d1, d2,
232           output logic[31:0] d3, d4);
233
234     always_ff @(posedge clk)
235     begin
236     if(reset)begin
237     d3 <= 32'b0;
238     d4 <= 32'b0;
```



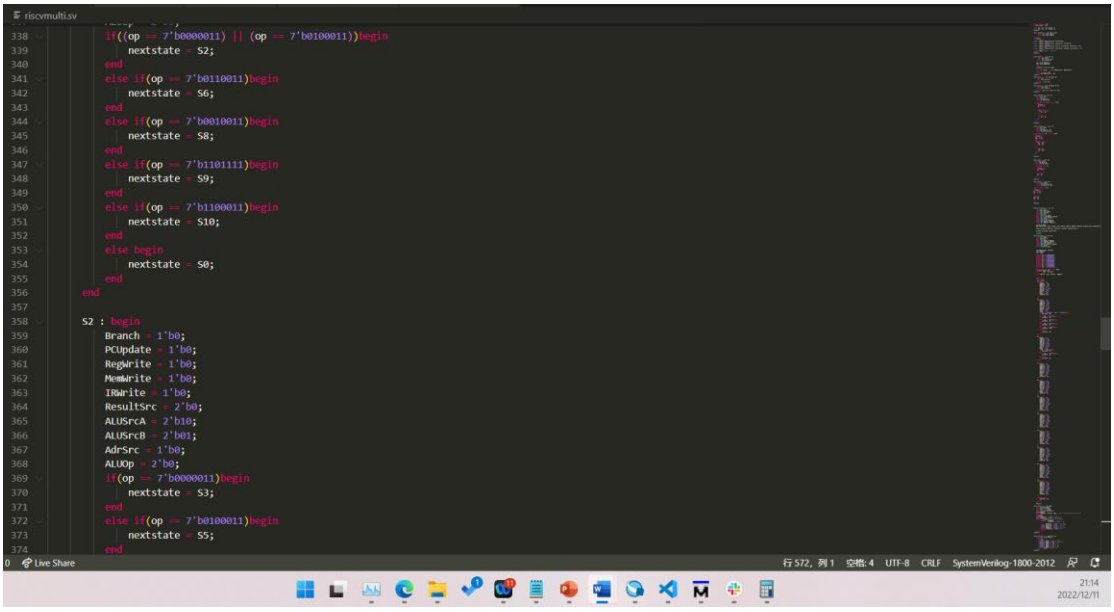
```
riscvmulti.sv
235     begin
236     if(reset)begin
237     d3 <= 32'b0;
238     d4 <= 32'b0;
239     end
240
241     else begin
242     d3 <= d1;
243     d4 <= d2;
244     end
245
246     end
247     endmodule
248
249
250
251     module controller(input logic clk,
252         input logic reset,
253         input logic [6:0] op,
254         input logic [2:0] funct3,
255         input logic funct7b5,
256         input logic Zero,
257         output logic [1:0] immsrc,
258         output logic [1:0] ALUSrcA, ALUSrcB,
259         output logic [1:0] ResultSrc,
260         output logic AdrSrc,
261         output logic [2:0] ALUControl,
262         output logic IRWrite, PCWrite,
263         output logic RegWrite, MemWrite);
264
265         logic[1:0] ALUOp;
266         Main_FSM mfsm(clk, reset, op[6:0], Zero, PCWrite, IRWrite, RegWrite, MemWrite, ResultSrc[1:0], ALUSrcA[1:0], ALUSrcB[1:0], AdrSrc, ALUOp[1:0]);
267
268         aludec alu(op[5], ALUOp[1:0], funct3[2:0], funct7b5, ALUControl[2:0]);
269
270         instrdec ins(op[6:0], immsrc[1:0]);
271
```



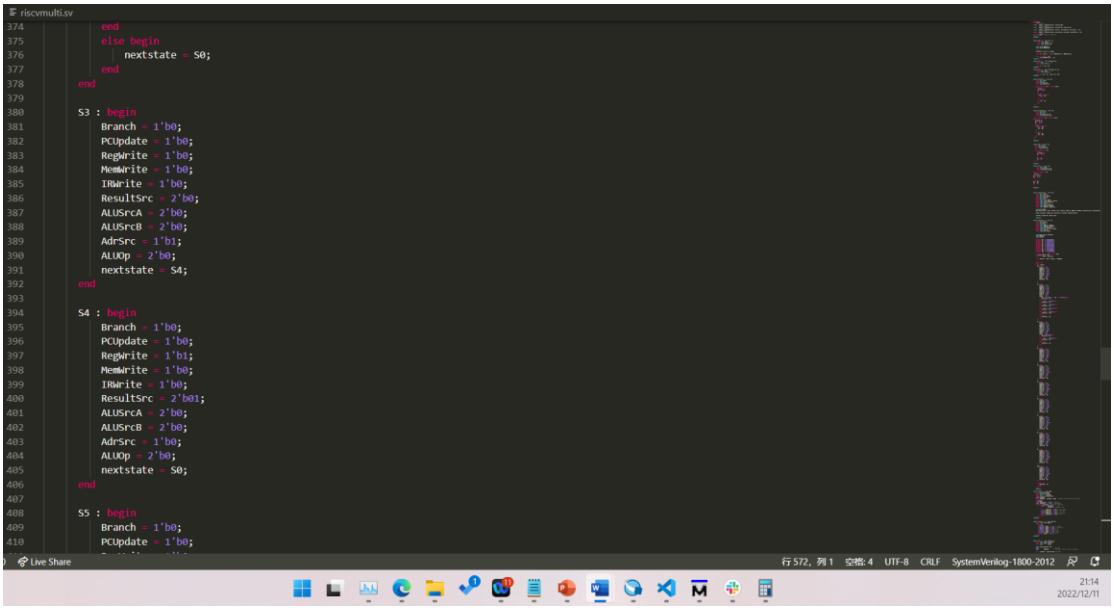
```
riscvmulti.sv
268         aludec alu(op[5], ALUOp[1:0], funct3[2:0], funct7b5, ALUControl[2:0]);
269
270         instrdec ins(op[6:0], immsrc[1:0]);
271
272         endmodule
273
274     module Main_FSM(input logic clk,
275         input logic reset,
276         input logic[6:0] op,
277         input logic Zero,
278         output logic PCWrite, IRWrite,
279         output logic RegWrite, MemWrite,
280         output logic[1:0] ResultSrc,
281         output logic[1:0] ALUSrcA, ALUSrcB,
282         output logic AdrSrc,
283         output logic[1:0] ALUOp);
284
285
286         logic[10:0] state, nextstate;
287         logic PCUpdate;
288         logic Branch;
289
290         parameter S0 = 11'b00000000001;
291         parameter S1 = 11'b00000000010;
292         parameter S2 = 11'b00000000100;
293         parameter S3 = 11'b00000001000;
294         parameter S4 = 11'b00000010000;
295         parameter S5 = 11'b00000100000;
296         parameter S6 = 11'b00001000000;
297         parameter S7 = 11'b00010000000;
298         parameter S8 = 11'b00100000000;
299         parameter S9 = 11'b01000000000;
300         parameter S10 = 11'b10000000000;
301
302         always_ff @(posedge clk, posedge reset)
303         if (reset) state = S0;
304         else       state = nextstate;
```

riscvmulti.sv

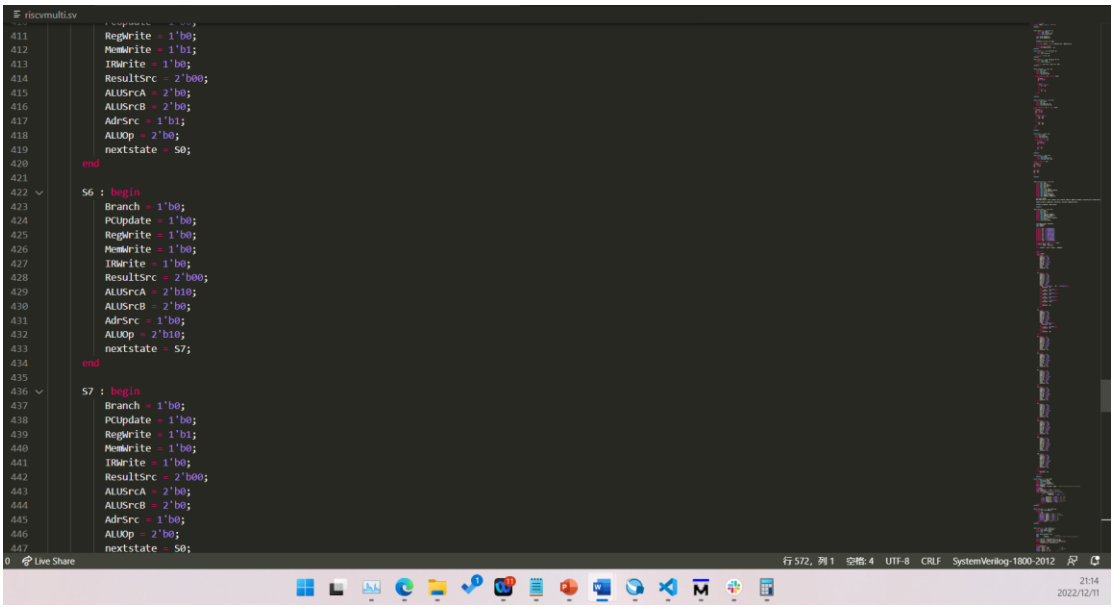
```
301
302     always_ff @(posedge clk, posedge reset)
303     if (reset) state = S0;
304     else       state = nextstate;
305
306     assign PCWrite = (Zero & Branch) | PCUpdate;
307
308
309     always_comb
310     begin
311       case (state)
312       S0 : begin
313           Branch = 1'b0;
314           PCUpdate = 1'b1;
315           RegWrite = 1'b0;
316           MemWrite = 1'b0;
317           IRWrite = 1'b1;
318           ResultSrc = 2'b10;
319           ALUSrcA = 2'b0;
320           ALUSrcB = 2'b10;
321           AdrSrc = 1'b0;
322           ALUOp = 2'b00;
323           nextstate = S1;
324
325       end
326
327       S1 : begin
328           Branch = 1'b0;
329           PCUpdate = 1'b0;
330           RegWrite = 1'b0;
331           MemWrite = 1'b0;
332           IRWrite = 1'b0;
333           ResultSrc = 2'b0;
334           ALUSrcA = 2'b01;
335           ALUSrcB = 2'b01;
336           AdrSrc = 1'b0;
337           ALUOp = 2'b0;
```

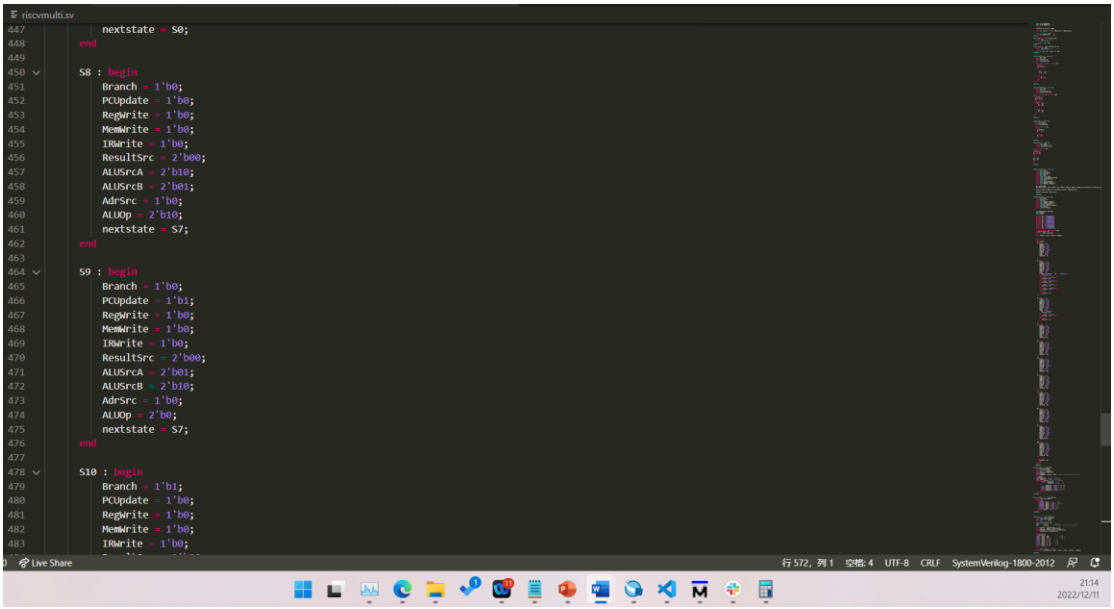



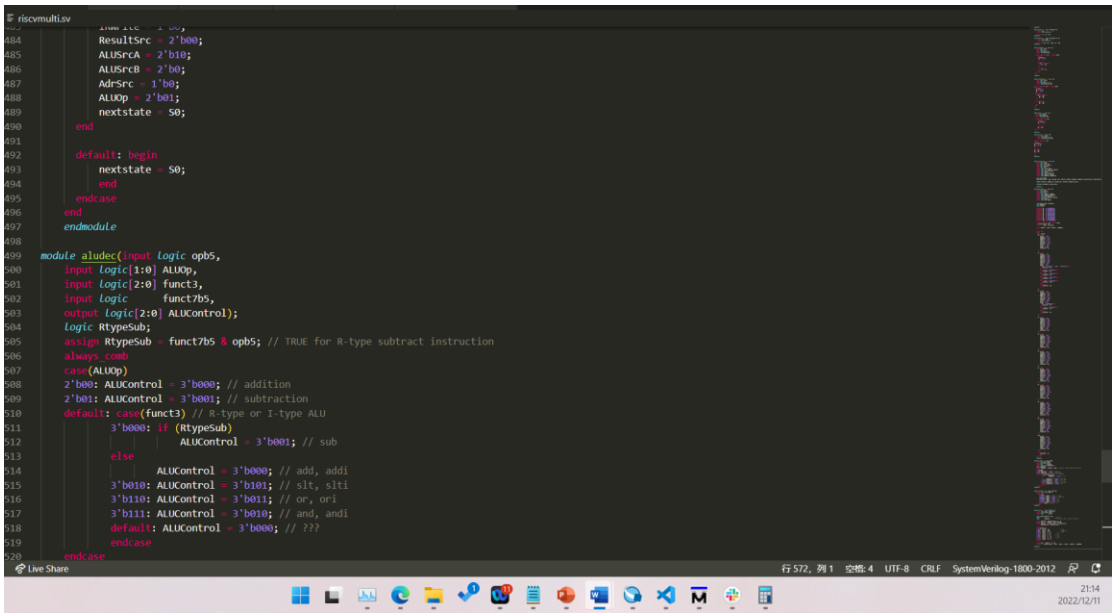
```
338           if((op == 7'b0000011) || (op == 7'b0100011))begin
339               nextstate = S2;
340           end
341           else if(op == 7'b0110011)begin
342               nextstate = S6;
343           end
344           else if(op == 7'b0010011)begin
345               nextstate = S8;
346           end
347           else if(op == 7'b1101111)begin
348               nextstate = S9;
349           end
350           else if(op == 7'b1100011)begin
351               nextstate = S10;
352           end
353           else begin
354               nextstate = S0;
355           end
356       end
357
358       S2 : begin
359           Branch = 1'b0;
360           PCUpdate = 1'b0;
361           RegWrite = 1'b0;
362           MemWrite = 1'b0;
363           IRWrite = 1'b0;
364           ResultSrc = 2'b0;
365           ALUSrcA = 2'b10;
366           ALUSrcB = 2'b01;
367           AdrSrc = 1'b0;
368           ALUOp = 2'b0;
369           if(op == 7'b0000011)begin
370               nextstate = S3;
371           end
372           else if(op == 7'b0100011)begin
373               nextstate = S5;
374           end
```

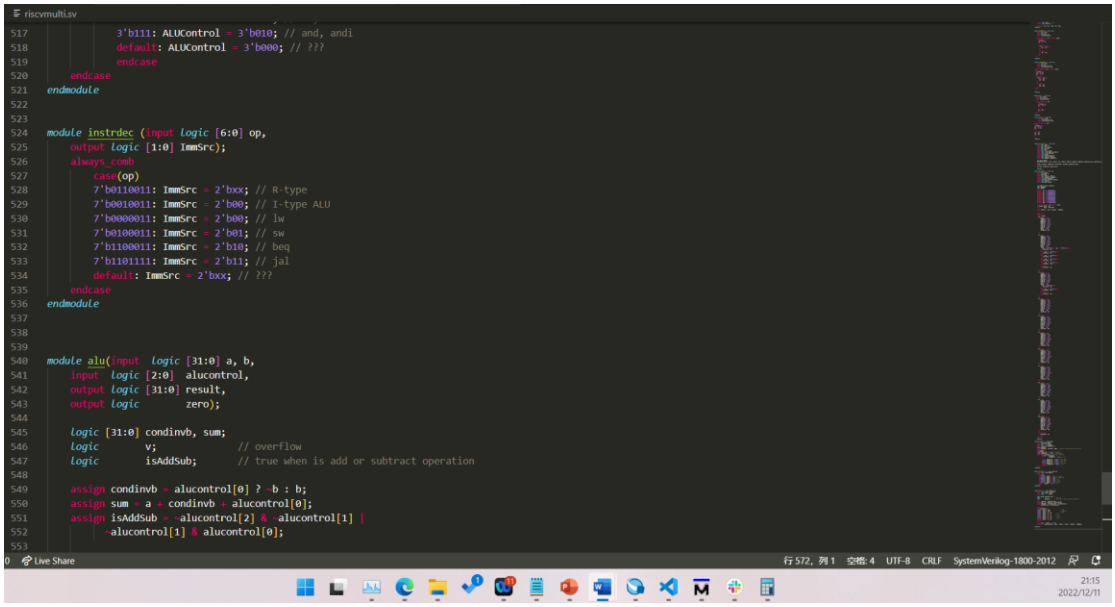



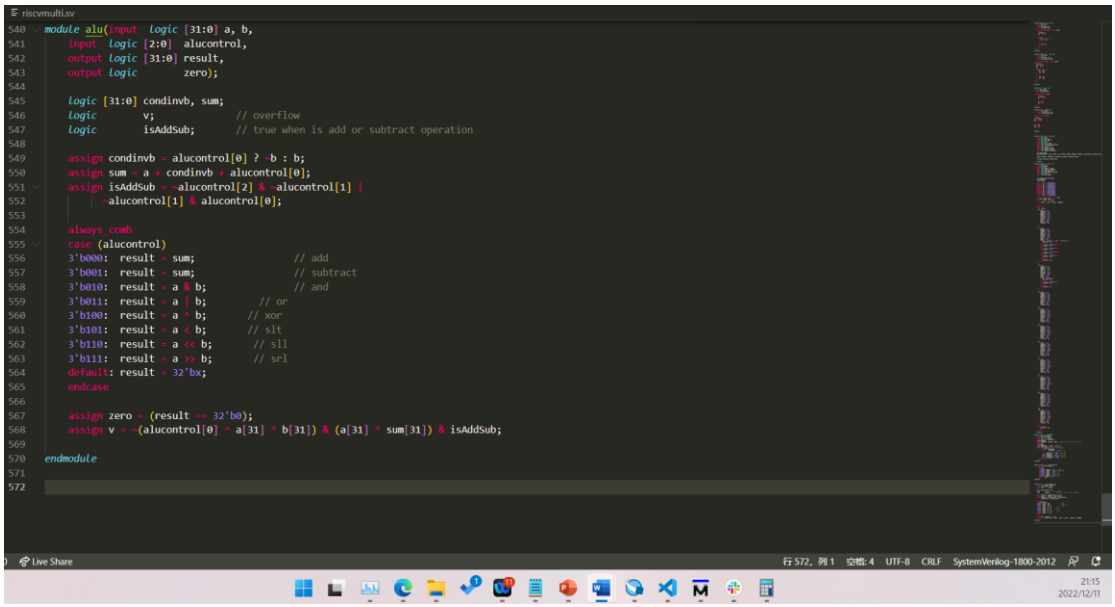
```
374           end
375           else begin
376               nextstate = S0;
377           end
378       end
379
380       S3 : begin
381           Branch = 1'b0;
382           PCUpdate = 1'b0;
383           RegWrite = 1'b0;
384           MemWrite = 1'b0;
385           IRWrite = 1'b0;
386           ResultSrc = 2'b0;
387           ALUSrcA = 2'b0;
388           ALUSrcB = 2'b0;
389           AdrSrc = 1'b1;
390           ALUOp = 2'b0;
391           nextstate = S4;
392       end
393
394       S4 : begin
395           Branch = 1'b0;
396           PCUpdate = 1'b0;
397           RegWrite = 1'b1;
398           MemWrite = 1'b0;
399           IRWrite = 1'b0;
400           ResultSrc = 2'b01;
401           ALUSrcA = 2'b0;
402           ALUSrcB = 2'b0;
403           AdrSrc = 1'b0;
404           ALUOp = 2'b0;
405           nextstate = S0;
406       end
407
408       S5 : begin
409           Branch = 1'b0;
410           PCUpdate = 1'b0;
```


riscvmulti.sv
RegWrite = 1'b0;
MemWrite = 1'b1;
IRWrite = 1'b0;
ResultSrc = 2'b00;
ALUSrcA = 2'b0;
ALUSrcB = 2'b0;
AdrSrc = 1'b1;
ALUOp = 2'b0;
nextstate = S0;
end
S6 : begin
Branch = 1'b0;
PCUpdate = 1'b0;
RegWrite = 1'b0;
MemWrite = 1'b0;
IRWrite = 1'b0;
ResultSrc = 2'b00;
ALUSrcA = 2'b10;
ALUSrcB = 2'b0;
AdrSrc = 1'b0;
ALUOp = 2'b10;
nextstate = S7;
end
S7 : begin
Branch = 1'b0;
PCUpdate = 1'b0;
RegWrite = 1'b1;
MemWrite = 1'b0;
IRWrite = 1'b0;
ResultSrc = 2'b00;
ALUSrcA = 2'b0;
ALUSrcB = 2'b0;
AdrSrc = 1'b0;
ALUOp = 2'b0;
nextstate = S0;
Live Share
行 572，列 1
空格: 4
UTF-8
CRLF
SystemVerilog-1800-2012
21:14
2022/12/11


riscvmulti.sv
nextstate = S0;
end
S8 : begin
Branch = 1'b0;
PCUpdate = 1'b0;
RegWrite = 1'b0;
MemWrite = 1'b0;
IRWrite = 1'b0;
ResultSrc = 2'b00;
ALUSrcA = 2'b10;
ALUSrcB = 2'b01;
AdrSrc = 1'b0;
ALUOp = 2'b10;
nextstate = S7;
end
S9 : begin
Branch = 1'b0;
PCUpdate = 1'b1;
RegWrite = 1'b0;
MemWrite = 1'b0;
IRWrite = 1'b0;
ResultSrc = 2'b00;
ALUSrcA = 2'b01;
ALUSrcB = 2'b10;
AdrSrc = 1'b0;
ALUOp = 2'b0;
nextstate = S7;
end
S10 : begin
Branch = 1'b1;
PCUpdate = 1'b0;
RegWrite = 1'b0;
MemWrite = 1'b0;
IRWrite = 1'b0;
Live Share
行 572，列 1
空格: 4
UTF-8
CRLF
SystemVerilog-1800-2012
21:14
2022/12/11


riscvmulti.sv
ResultSrc = 2'b00;
ALUSrcA = 2'b10;
ALUSrcB = 2'b0;
AdrSrc = 1'b0;
ALUOp = 2'b01;
nextstate = S0;
default: begin
nextstate = S0;
endcase
endmodule
module aludec(input logic opb5,
input logic[1:0] ALUOp,
input logic[2:0] funct3,
input logic funct7b5,
output logic[2:0] ALUControl);
logic RtypeSub;
assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract instruction
always_comb
case(ALUOp)
2'b00: ALUControl = 3'b000; // addition
2'b01: ALUControl = 3'b001; // subtraction
default: case(funct3) // R-type or I-type ALU
3'b000: if (RtypeSub)
ALUControl = 3'b001; // sub
else
ALUControl = 3'b000; // add, addi
3'b010: ALUControl = 3'b101; // slt, slti
3'b110: ALUControl = 3'b011; // or, ori
3'b111: ALUControl = 3'b010; // and, andi
default: ALUControl = 3'b000; // ???
endcase
endcase
Live Share
行 572, 列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012
21:14
2022/12/11


riscvmulti.sv
3'b111: ALUControl = 3'b010; // and, andi
default: ALUControl = 3'b000; // ???
endcase
endcase
endmodule
module instrdec (input logic [6:0] op,
output logic [1:0] ImmSrc);
always_comb
case(op)
7'b0110011: ImmSrc = 2'bxx; // R-type
7'b0010011: ImmSrc = 2'b00; // I-type ALU
7'b0000011: ImmSrc = 2'b00; // lw
7'b0100011: ImmSrc = 2'b01; // sw
7'b1100011: ImmSrc = 2'b10; // beq
7'b1101111: ImmSrc = 2'b11; // jal
default: ImmSrc = 2'bxx; // ???
endcase
endmodule
module alu(input logic [31:0] a, b,
input logic [2:0] alucontrol,
output logic [31:0] result,
output logic zero);
logic [31:0] condinvb, sum;
logic v; // overflow
logic isAddSub; // true when is add or subtract operation
assign condinvb = alucontrol[0] ? ~b : b;
assign sum = a + condinvb + alucontrol[0];
assign isAddSub = ~alucontrol[2] & ~alucontrol[1] |
~alucontrol[1] & alucontrol[0];
Live Share
行 572, 列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012
21:15
2022/12/11


riscvmulti.sv
module alu(input logic [31:0] a, b,
input logic [2:0] alucontrol,
output logic [31:0] result,
output logic zero);
logic [31:0] condinvb, sum;
logic v; // overflow
logic isAddSub; // true when is add or subtract operation
assign condinvb = alucontrol[0] ? ~b : b;
assign sum = a + condinvb + alucontrol[0];
assign isAddSub = ~alucontrol[2] & ~alucontrol[1] |
~alucontrol[1] & alucontrol[0];
always_comb
case (alucontrol)
3'b000: result = sum; // add
3'b001: result = sum; // subtract
3'b010: result = a & b; // and
3'b011: result = a | b; // or
3'b100: result = a ^ b; // xor
3'b101: result = a < b; // slt
3'b110: result = a << b; // sll
3'b111: result = a >> b; // srl
default: result = 32'bx;
endcase
assign zero = (result == 32'b0);
assign v = ~(alucontrol[0] ^ a[31] ^ b[31]) & (a[31] ^ sum[31]) & isAddSub;
endmodule
Live Share
行 572, 列 1 空格: 4 UTF-8 CRLF SystemVerilog-1800-2012
21:15
2022/12/11

4.

Table 1: Expected Operation (after two cycles of reset)

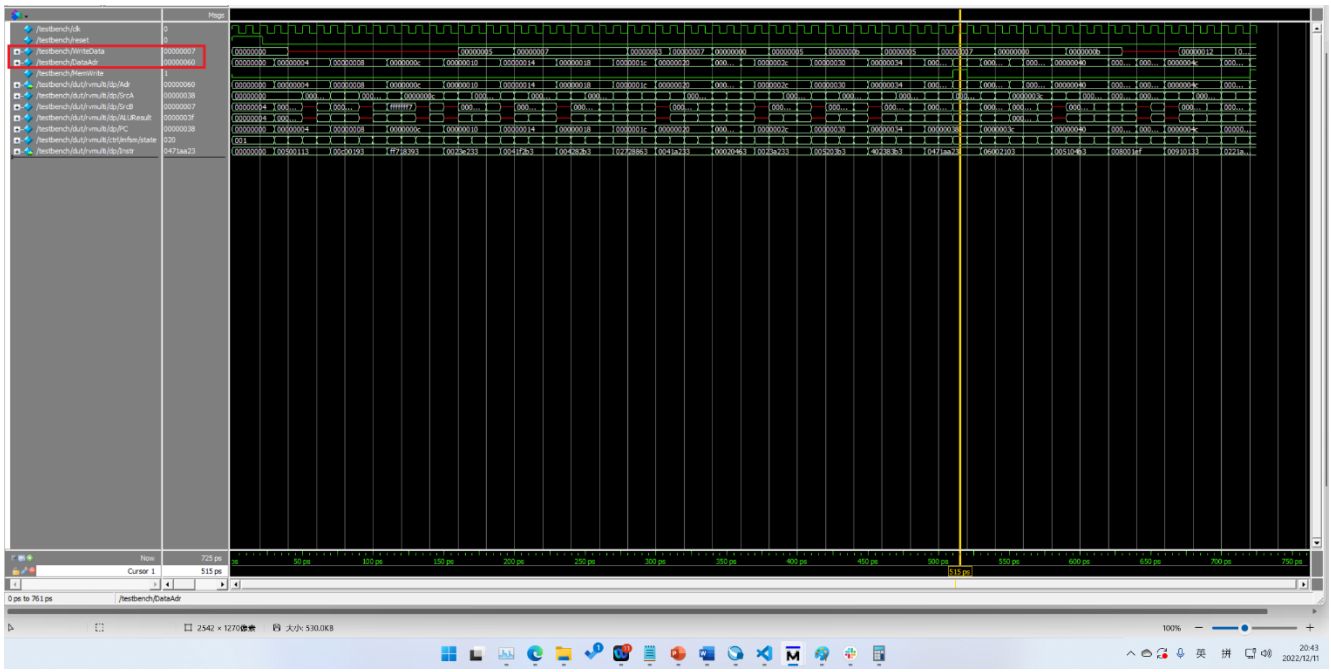
Step	PC	Instr	State	Result	Result Notes
3	00	00500113	S0: Fetch	4	PC+4
4	04	""	S1: Decode	X	OldPC+Immediate
5	04	""	S8: ExecuteI	X	ALUResult = x0 (0) + 5 = 5
6	04	""	S7: ALUWB	5	Result = ALUOUT
7	04	""	S0: Fetch	8	PC+4
8	08	00c00193	S1: Decode	X	OldPC+Immediate
9	08	""	S8: ExecuteI	X	ALUResult = x0(0)+12=12
10	08	""	S7:ALUWB	12	Result = ALUOUT
11	08	""	S0:Fetch	12	PC+4
12	12	FF718393	S1:Decode	X	OldPC+Immediate
13	12	""	S8:ExecuteI	X	ALUResult = x3(12)-9=3
14	12	""	S7:ALUWB	3	Result = ALUOUT
15	12	""	S0:Fetch	16	PC+4
16					
17					
18					
19					
20					

5.

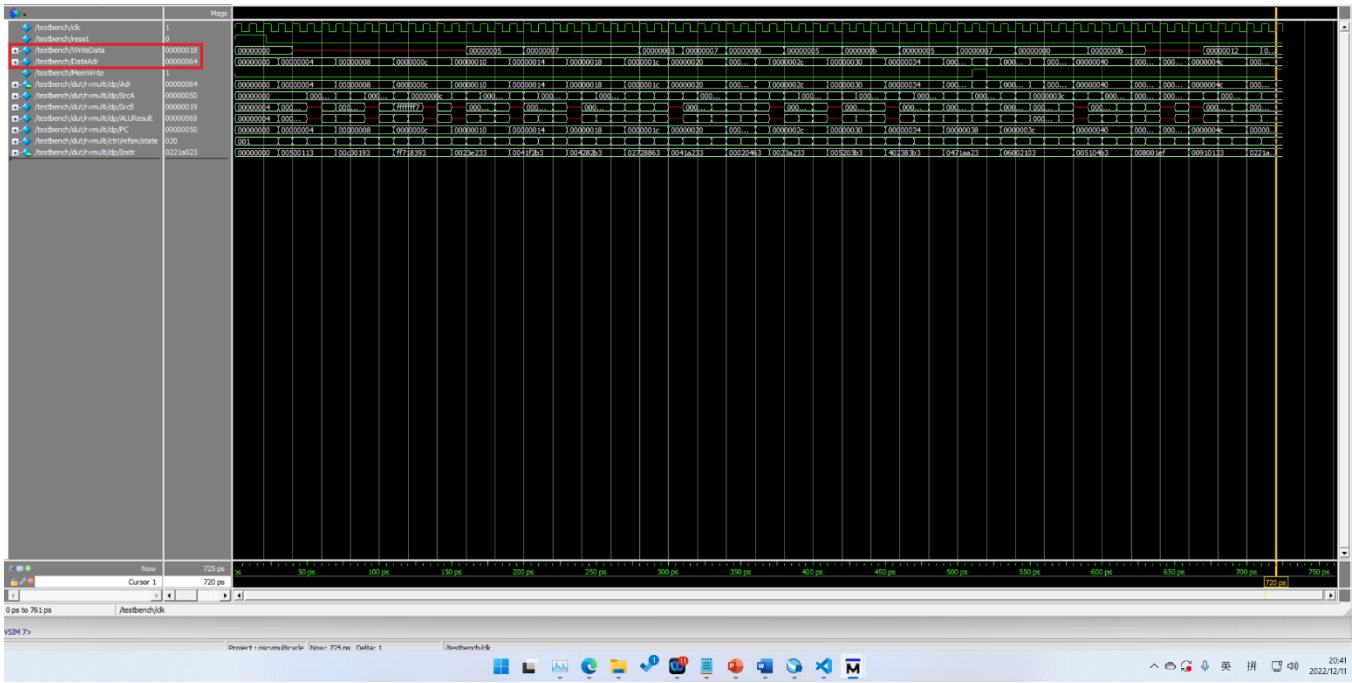
5.

5.

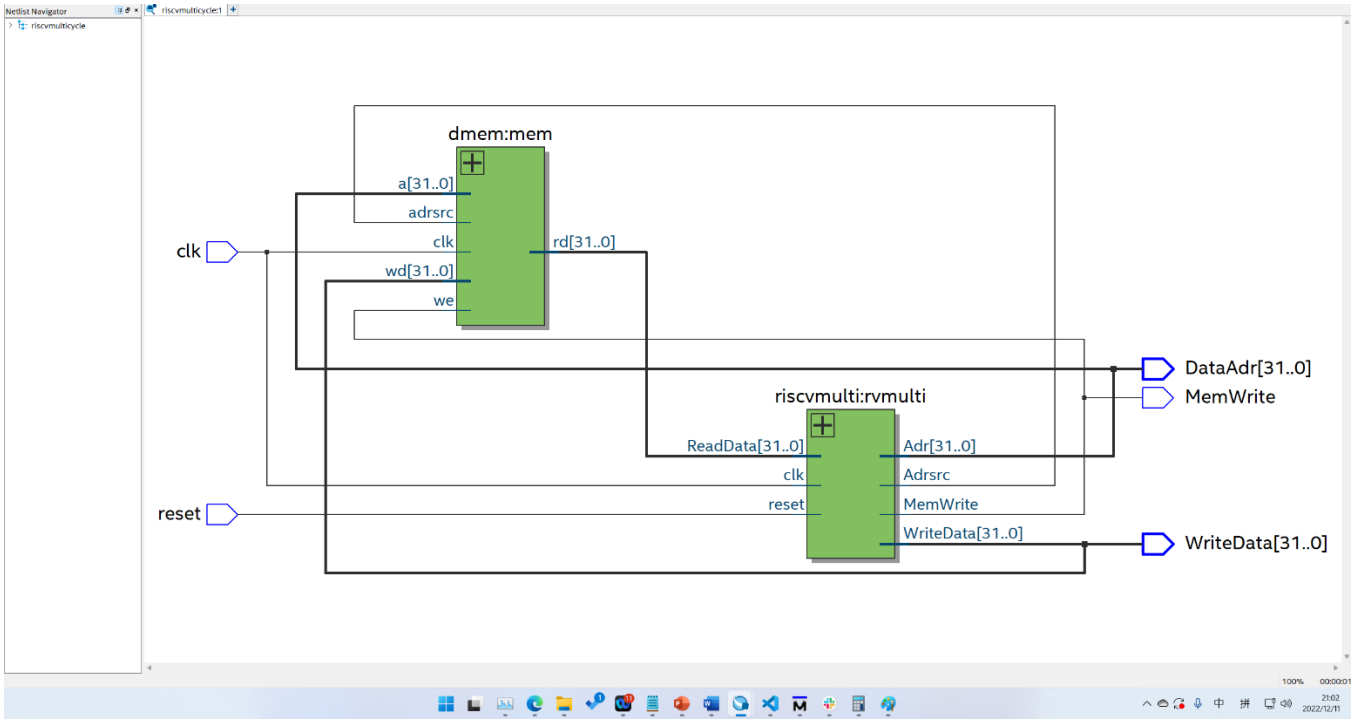
WriteData is 7. Writes into 60 in hex, which is 96 in dec.

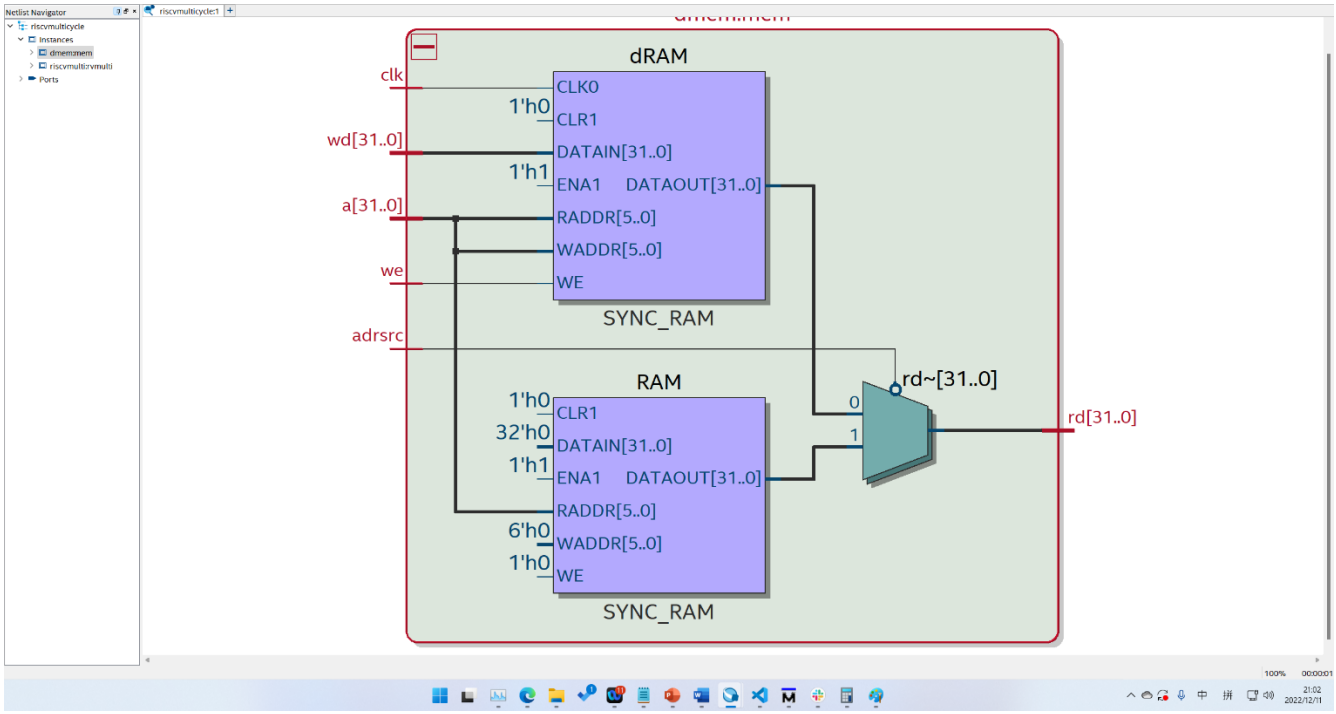


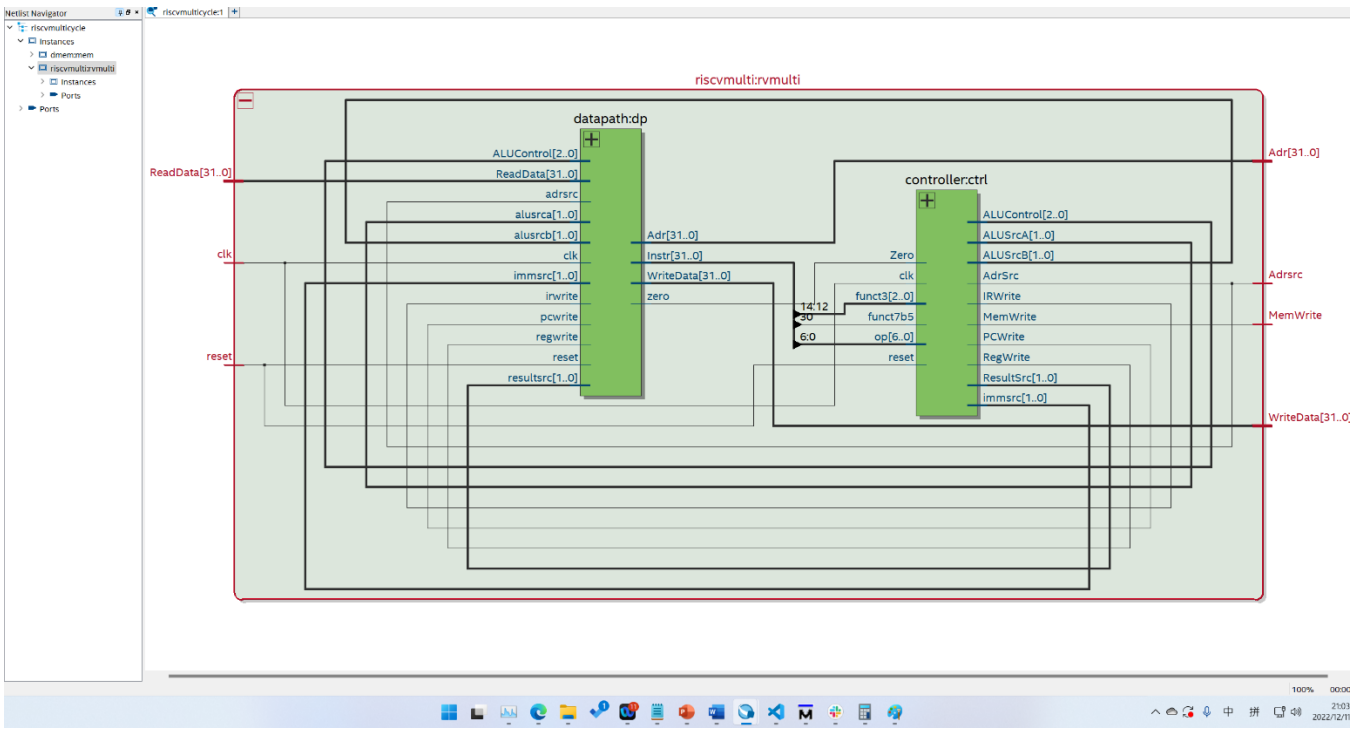
WriteData is 19 in hex, which is 25 in dec. Writes into 64 in hex, which is 100 in dec.

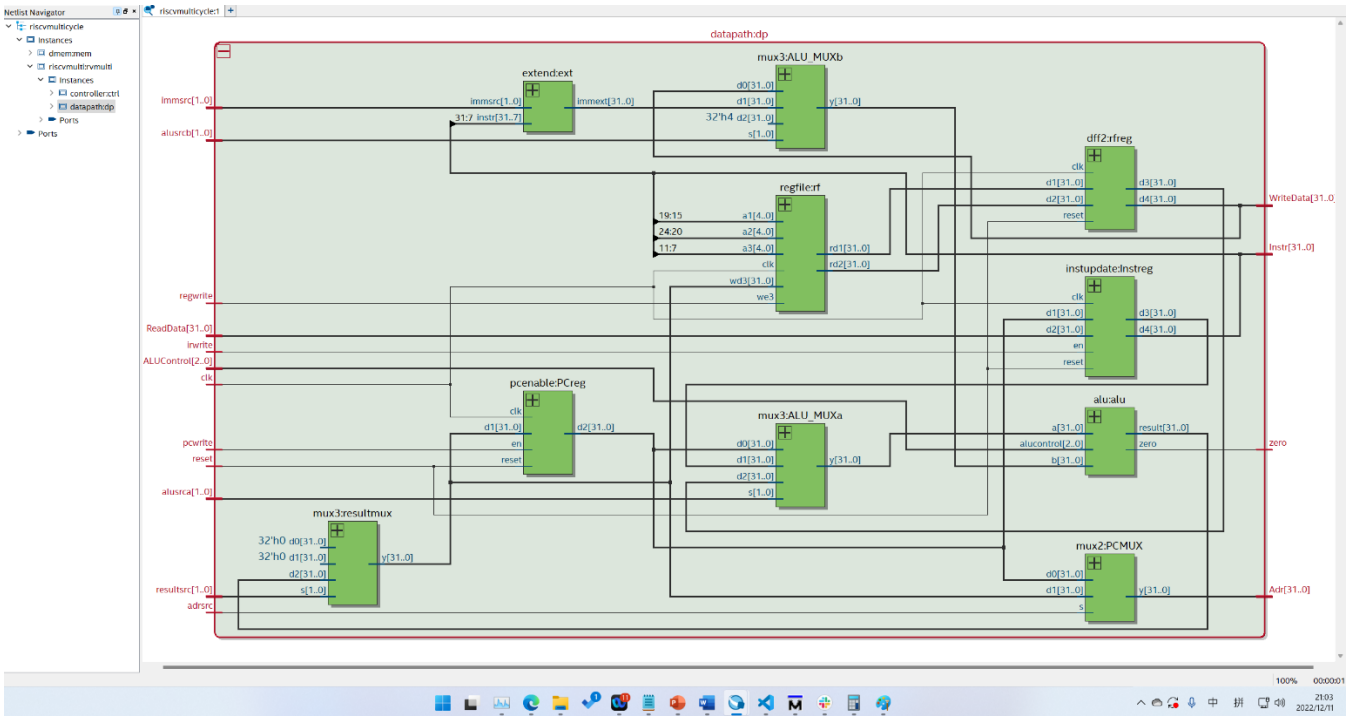


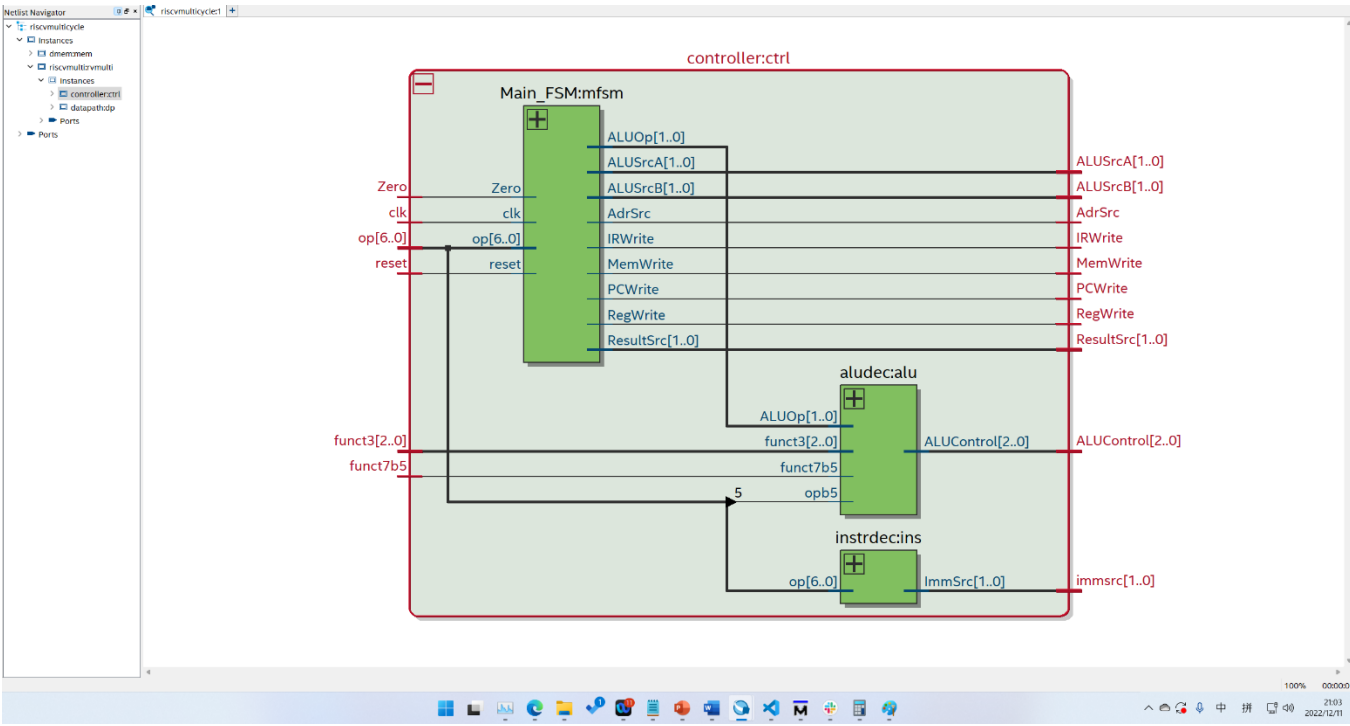
8.

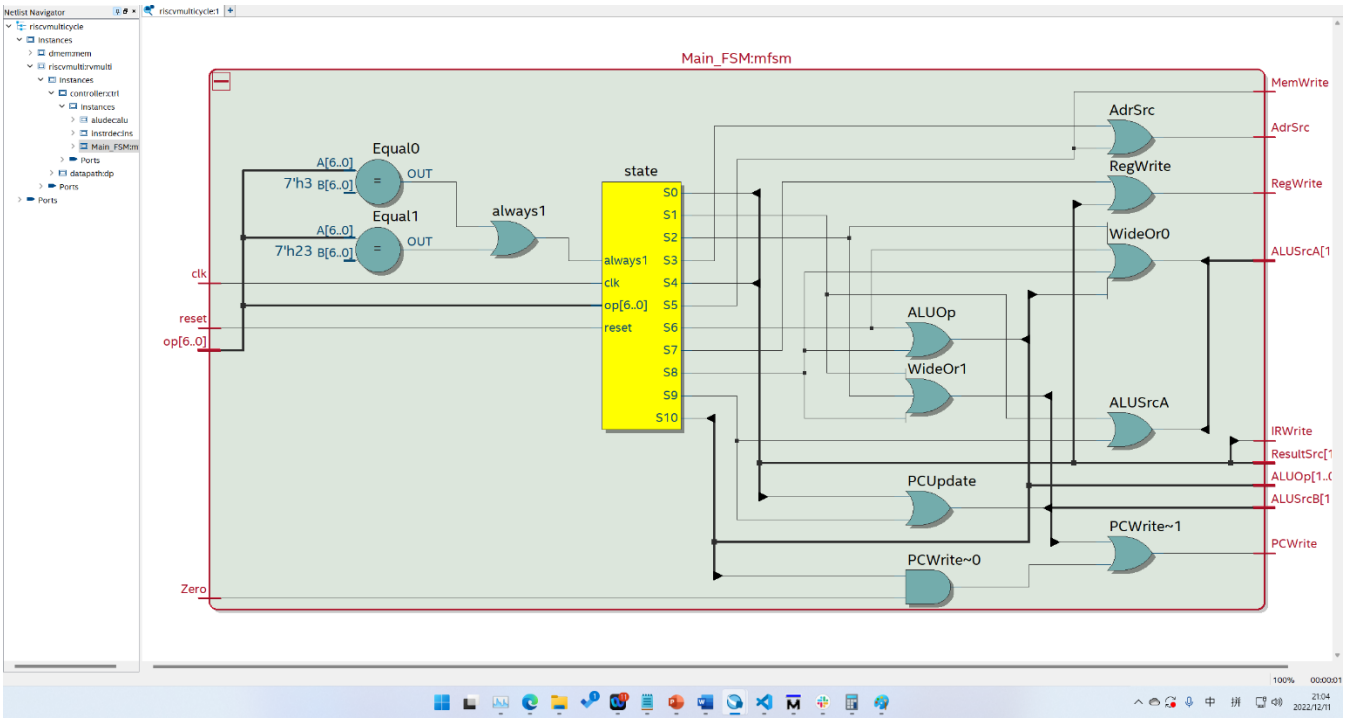

dmem:mem
a[31..0]
adrsrc
clk
wd[31..0]
we
rd[31..0]
clk
reset
riscvmulti:rvmulti
ReadData[31..0]
clk
reset
Adr[31..0]
Adrsrc
MemWrite
WriteData[31..0]
DataAdr[31..0]
MemWrite
WriteData[31..0]


dRAM
clk
1'h0
wd[31..0]
1'h1
a[31..0]
we
adrsrc
CLK0
CLR1
DATAIN[31..0]
ENA1
DATAOUT[31..0]
RADDR[5..0]
WADDR[5..0]
WE
SYNC_RAM
RAM
1'h0
32'h0
1'h1
6'h0
1'h0
CLR1
DATAIN[31..0]
ENA1
DATAOUT[31..0]
RADDR[5..0]
WADDR[5..0]
WE
SYNC_RAM
rd~[31..0]
0
1
rd[31..0]


Netlist Navigator
riscvmulticycle
Instances
dmemmem
riscvmultirvmulti
Instances
Ports
Ports
riscvmulticycle:1
riscvmulti:rvmulti
datapath:dp
ALUControl[2..0]
ReadData[31..0]
adrsrc
alusrca[1..0]
alusrcb[1..0]
clk
immsrc[1..0]
irwrite
pcwrite
regwrite
reset
resultsrc[1..0]
Adr[31..0]
Instr[31..0]
WriteData[31..0]
zero
controller:ctrl
Zero
clk
funct3[2..0]
funct7b5
op[6..0]
reset
ALUControl[2..0]
ALUSrcA[1..0]
ALUSrcB[1..0]
AdrSrc
IRWrite
MemWrite
PCWrite
RegWrite
ResultSrc[1..0]
immsrc[1..0]
14:12
30
6:0
Adr[31..0]
Adrsrc
MemWrite
WriteData[31..0]
100%
00:00
21:03
2022/12/11


Netlist Navigator
riscvmulticycle
Instances
dmemmem
riscvmultirvmulti
Instances
controller:ctrl
datapath:dp
Ports
Ports
riscvmulticycle:1
datapath:dp
extend:ext
immsrc[1..0]
31:7 instr[31..7]
immext[31..0]
mux3:ALU_MUXb
d0[31..0]
d1[31..0]
32'h4 d2[31..0]
s[1..0]
y[31..0]
regfile:rf
19:15
24:20
11:7
a1[4..0]
a2[4..0]
a3[4..0]
clk
wd3[31..0]
we3
rd1[31..0]
rd2[31..0]
dff2:rfreg
clk
d1[31..0]
d2[31..0]
reset
d3[31..0]
d4[31..0]
WriteData[31..0]
Instr[31..0]
instupdate:Instreg
clk
d1[31..0]
d2[31..0]
en
reset
d3[31..0]
d4[31..0]
immsrc[1..0]
alusrcb[1..0]
regwrite
ReadData[31..0]
irwrite
ALUControl[2..0]
clk
pcwrite
reset
alusrca[1..0]
resultsrc[1..0]
adrsrc
pcenable:PCreg
clk
d1[31..0]
en
reset
d2[31..0]
mux3:ALU_MUXa
d0[31..0]
d1[31..0]
d2[31..0]
s[1..0]
y[31..0]
alu:alu
a[31..0]
alucontrol[2..0]
b[31..0]
result[31..0]
zero
mux3:resultmux
32'h0 d0[31..0]
32'h0 d1[31..0]
d2[31..0]
s[1..0]
y[31..0]
mux2:PCMUX
d0[31..0]
d1[31..0]
s
y[31..0]
Adr[31..0]
100%
00:00:01
21:03
2022/12/11


Netlist Navigator
riscvmulticycle:1
controller:ctrl
Main_FSM:mfsm
ALUOp[1..0]
ALUSrcA[1..0]
ALUSrcB[1..0]
AdrSrc
IRWrite
MemWrite
PCWrite
RegWrite
ResultSrc[1..0]
Zero
clk
op[6..0]
reset
aludec:alu
funct3[2..0]
funct7b5
opb5
ALUControl[2..0]
instrdec:ins
ImmSrc[1..0]
immsrc[1..0]
5


Netlist Navigator
riscvmulticycle:1
Main_FSM:mfsm
Equal0
Equal1
A[6..0]
7'h3
7'h23
B[6..0]
OUT
always1
state
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
clk
reset
op[6..0]
MemWrite
AdrSrc
RegWrite
WideOr0
ALUSrcA[1
ALUOp
WideOr1
ALUSrcA
IRWrite
ResultSrc[1
ALUOp[1..0
ALUSrcB[1
PCUpdate
PCWrite~1
PCWrite
PCWrite~0
Zero