DIGITAL DESIGN

LAB4 VERILOG(LOOP、PRIMITIVE、STRUCTURE DESIGN)

2022 FALL TERM @ CSE . SUSTECH

LAB4

- Sum of Minterms vs Product of Maxterms
- Verilog
  - loop in testbench: repeat, forever, for, while
  - primitive
  - structured design
- Practise

SUM-O-PRODUCT VS SUM-OF-MINTERMS

- F(A,B,C)=A+B'C
- F(A,B,C)=AB(C'+C)+AB'(C'+C)+B'C(A'+A)=A'B'C+AB'C'+AB'C+ABC'+ABC

  =m1+m4+m5+m6+m7=∑(1,4,5,6,7)

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Minterm	A	B	C	F
m0	0	0	0	0
m1	0	0	1	1
m2	0	1	0	0
m3	0	1	1	0
m4	1	0	0	1
m5	1	0	1	1
m6	1	1	0	1
m7	1	1	1	1

SUM-OF-MINTERMS VS PRODUCT-OF-MAXTERMS

- F(A,B,C)=A+B'C = AB(C'+C)+AB'(C'+C)+B'C(A'+A)=A'B'C+AB'C'+AB'C+ABC'+ABC

  =m1+m4+m5+m6+m7=∑(1,4,5,6,7)

Minterm	A	B	C	F	F'
m0	0	0	0	0	1
m1	0	0	1	1	0
m2	0	1	0	0	1
m3	0	1	1	0	1
m4	1	0	0	1	0
m5	1	0	1	1	0
m6	1	1	0	1	0
m7	1	1	1	1	0

F(A,B,C) = F(A,B,C)''

= (F(A,B,C)')'

= ((∑(1,4,5,6,7)')'

= (m0+m2+m3)'

= (A'B'C' + A'BC'+A'BC)'

= (A'B'C')' . (A'BC')' . (A'BC)'

= (A+B+C) . (A+B'+C) . (A+B'+C')

= M0.M2.M3 = ∏(0,2,3)

DESIGN IN DATAFLOW OF VERILOG

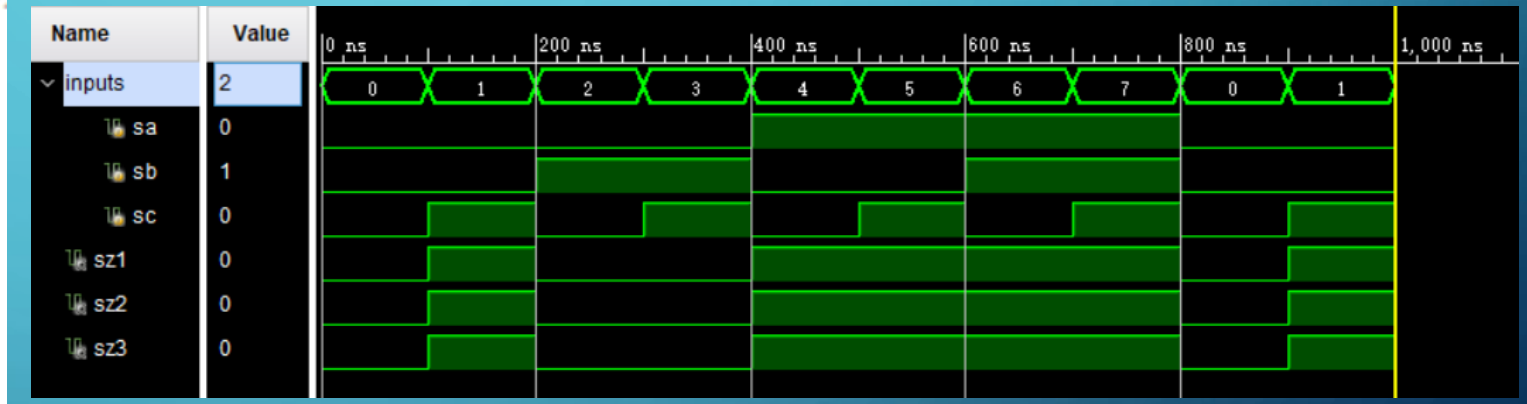
- z1=F(A,B,C)=A+B’C
- z2=F(A,B,C)=∑(1,4,5,6,7) =A’B’C+AB’C‘+AB’C+ABC’+ABC
- z3=F(A,B,C)= ∏(0,2,3)= (A+B+C) . (A+B’+C) . (A+B’+C’)

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

```
module sop_som_pom(input a, b, c, output z1, z2, z3);
    // a+b'c
    assign z1 = a | (~b & c);
    //m1(a'b'c) + m4(ab'c') + m5(ab'c) + m6(abc') + m7(abc)
    assign z2 = (~a & ~b & c) | (a & ~b & ~c) | (a & ~b & c) | (a & b &~c) | (a & b & c);
    //M0(a+b+c) . M2(a+b'+c) . M3(a+b'+c')
    assign z3 = (a | b | c) & (a | ~b | c) & (a | ~b | ~c);
endmodule
```

TESTBENCH USING LOOP(1)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module sop_som_pom_sim( );
reg sa, sb, sc;
wire sz1, sz2, sz3;
sop_som_pom u1(.a(sa), .b(sb), .c(sc), .z1(sz1), .z2(sz2), .z3(sz3));
initial begin
{sa, sb, sc} = 3'b0;
forever #100 {sa, sb, sc} = {sa, sb, sc} + 1;
end
endmodule
```



Tips on Vivado:

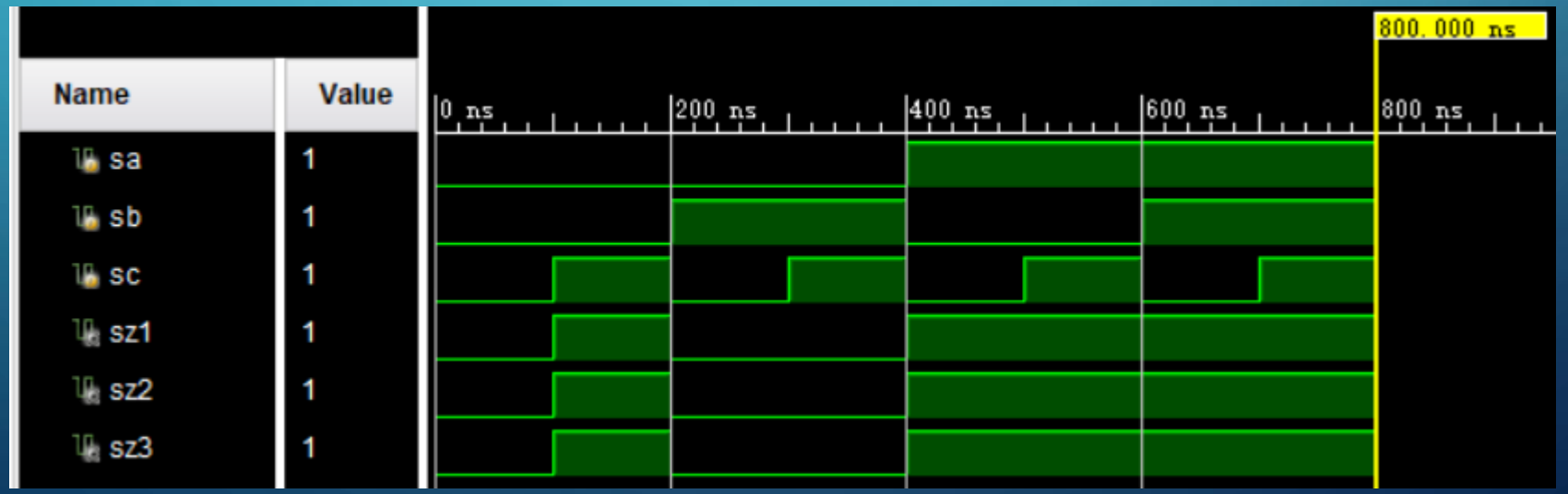
- Select the name of the ports you want to see in group (ctrl + select)
- Right click then choose the “New virtual Bus” to group all the selected ports, the name of the virtual bus could be edit

TESTBENCH USING LOOP(2)

```
initial begin
    {sa, sb, sc} = 3'b0;
    repeat(7) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #100 $finish();
end
```

```
initial begin
    {sa, sb, sc} = 3'b0;
    for(integer i=0;i<7;i=i+1) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #100 $finish();
end
```

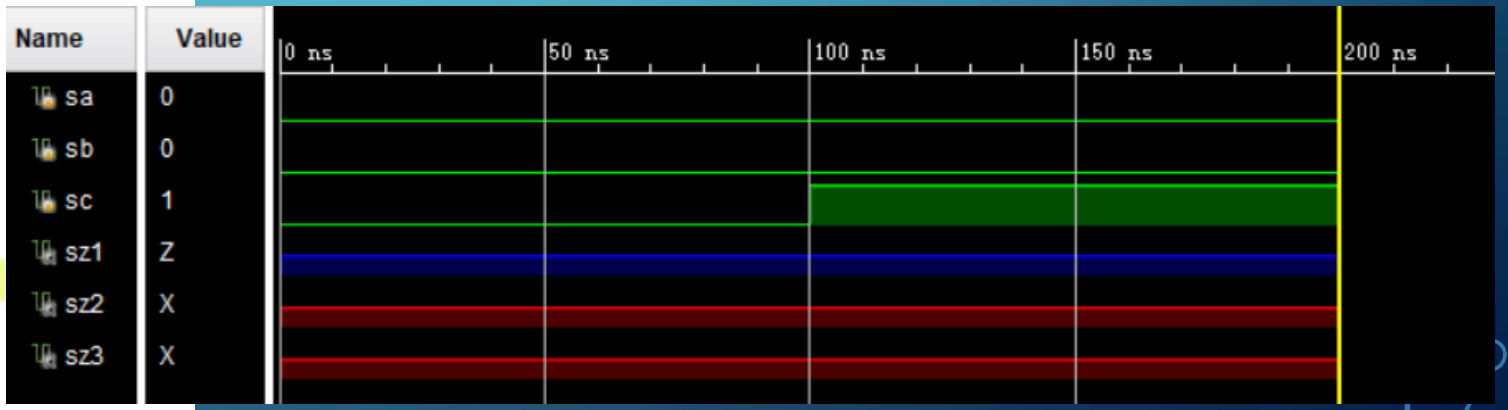
```
integer i=0;
initial begin
    {sa, sb, sc} = 3'b0;
    while(i<7) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
        i=i+1;
    end
    #100 $finish();
end
```



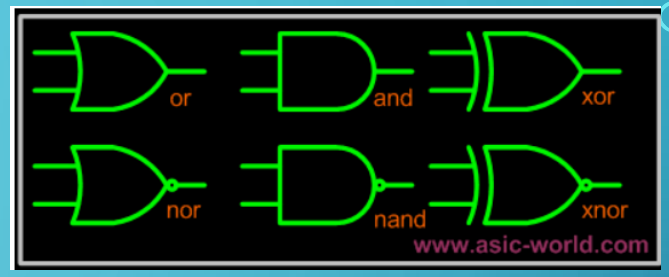
PRACTICES(1)

Do the simulation on sop_som_pom_sim which is the testbench of sop_som_pom, it is found that the waveform is not in line with expectations: 1) There are just two values of sa, sb and sc in the simulation, why ? 2) sz1 is Z, while the state of sz2 and sz3 is X, why?
Modify the testbench to make it workable and test the function of module sop_som_pom

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
21  module sop_som_pom_sim( );
22      reg sa, sb, sc;
23      wire sz1, sz2, sz3;
24      //sop_som_pom u1( .a(sa), .b(sb), .c(sc), .z1(sz1), .z2(sz2), .z3(sz3));
25      sop_som_pom u1(sa, sb, sz1, sz2, sz3);
26      /*...*/
36      initial begin
37          {sa, sb, sc} = 3'b0;
38          repeat(7): begin
39              #100 {sa, sb, sc} = {sa, sb, sc} + 1;
40          end
41          #100 $finish();
42      end
43    /*...*/
59  endmodule
```



PRIMITIVE



- Verilog has built in primitives like gates, transmission gates, and switches. These are rarely used in design (RTL Coding), but are used in post synthesis world for modeling the ASIC/FPGA cells; these cells are then used for gate level simulation, or what is called as SDF simulation. Also the output netlist format from the synthesis tool, which is imported into the place and route tool, is also in Verilog gate level primitives.
- Note : RTL engineers still may use gate level primitives or ASIC library cells in RTL when using IO CELLS, Cross domain synch cells.

http://www.asic-world.com/verilog/index.html

PRIMITIVE GATE(1)

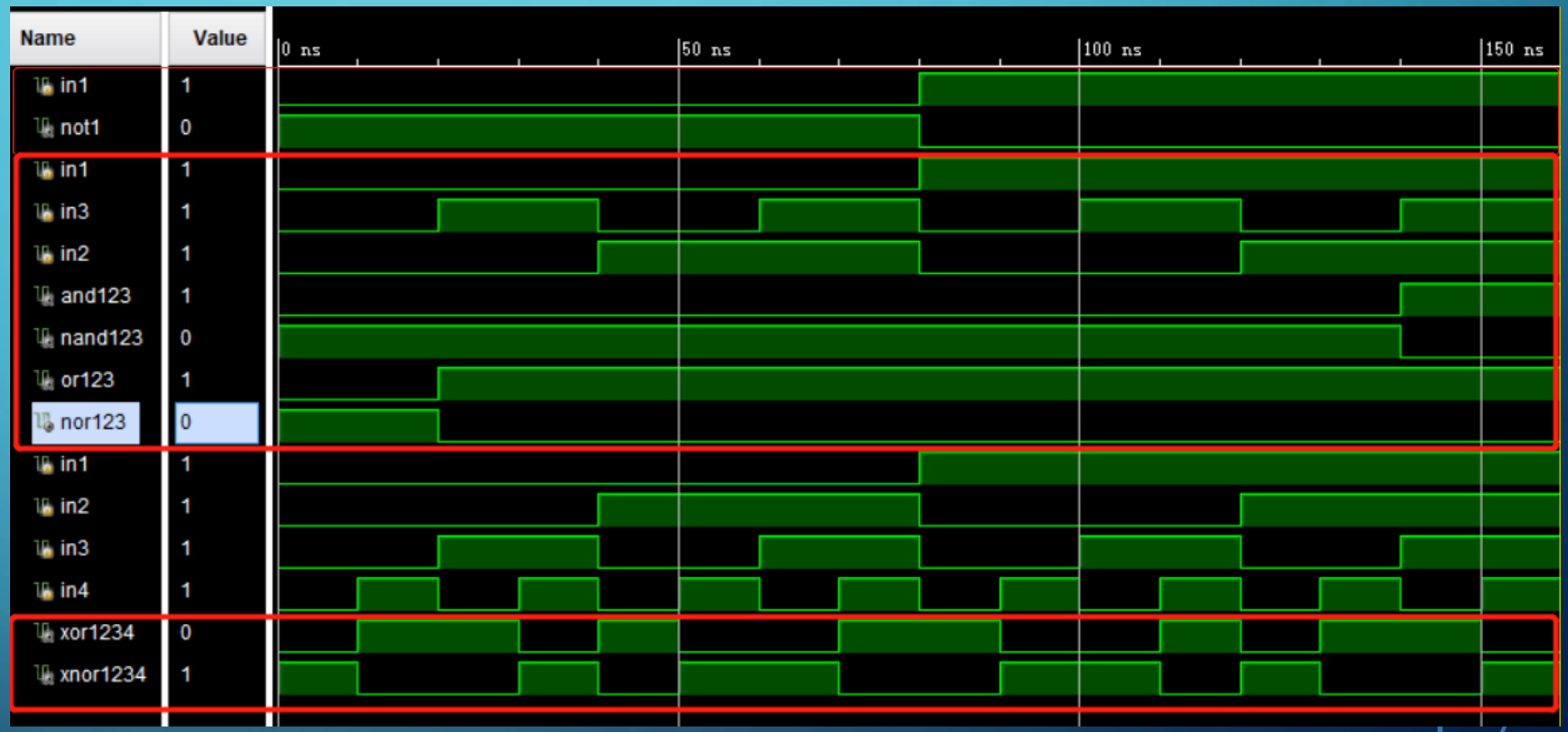
- The bitwise of all the primitive gates are 1bit.
- The 1st port is output, the left ports is(are) input.
- Only “not” gate has 1 input port, other primitive gates could has 1 or more than 1 inpurt ports.

Gate	Description
not	1-input 1-output NOT gate
and	N-input 1-output AND gate
nand	N-input 1-output NAND gate
or	N-input 1-output OR gate
nor	N-input 1-output NOR gate
xor	N-input 1-output XOR gate
xnor	N-input 1-output XNOR gate

```
module primitive_gates_sim( );
reg in1, in2, in3, in4;
wire not1, and123, nand123, or123, nor123, xor1234, xnor1234;

not notU1(not1, in1);
and andU1In3(and123, in1, in2, in3);
nand nandU1In3(nand123, in1, in2, in3);
or orU1In3(or123, in1, in2, in3);
nor norU1In3(nor123, in1, in2, in3);
xor xorU1In4(xor1234, in1, in2, in3, in4);
xnor xnorU1In4(xnor1234, in1, in2, in3, in4);

initial begin
    {in1, in2, in3, in4} = 4'b0;
    repeat(15) #10 {in1, in2, in3, in4} = {in1, in2, in3, in4} + 1;
    #10 $finish();
end
endmodule
```

PRIMITIVE GATE(2)

```
module primitive_gates_sim( );
reg in1, in2, in3, in4;
wire not1, and123, nand123, or123, nor123, xor1234, xnor1234;

not notU1(not1, in1);
and andU1In3(and123, in1, in2, in3);
nand nandU1In3(nand123, in1, in2, in3);
or orU1In3(or123, in1, in2, in3);
nor norU1In3(nor123, in1, in2, in3);
xor xorU1In4(xor1234, in1, in2, in3, in4);
xnor xnorU1In4(xnor1234, in1, in2, in3, in4);

initial begin
    {in1, in2, in3, in4} = 4'b0;
    repeat(15) #10 {in1, in2, in3, in4} = {in1, in2, in3, in4} + 1;
    #10 $finish();
end
endmodule
```



TWO WAYS TO DO THE DESIGN

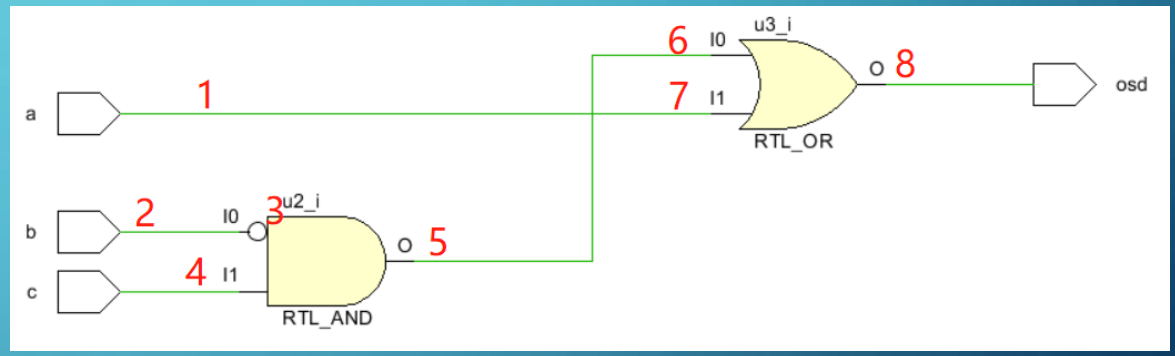
- Data flow design： using “assign “ as *continuous assignment* , to transfer the data from input ports through variables to the output ports .

- Structured design： using verilog to define the relationship between modules. It is based on the modules/primitive gates/IP cores.

STRUCTURE DESIGN IN VERILOG(1)

- Do the design in structure design by using the instances of the primitive gate.

```
module sop_som_pom_sd(input a, b, c, output osd);
    wire ou1, ou2;
    //f(a, b, c) = a+b'c.    assign odf = a | (~b & c);    // data-flow
    ////// structure-design //////
    not u1(ou1, b);       // ~b
    and u2(ou2, ou1, c); // ~b & c
    or u3(osd, ou2, a);  // a | (~b & c)
endmodule
```



Q: Please find signal “ou1” and “ou2” in the right figure

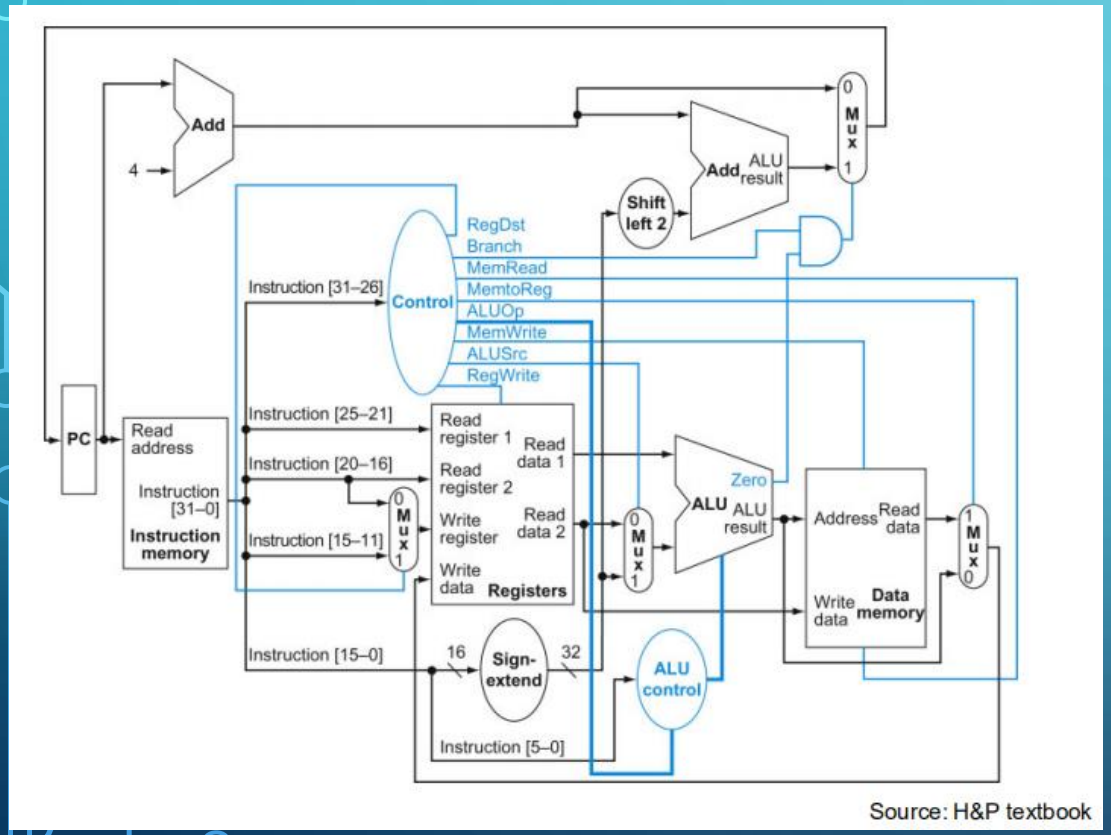
STRUCTURE DESIGN IN VERILOG(2)

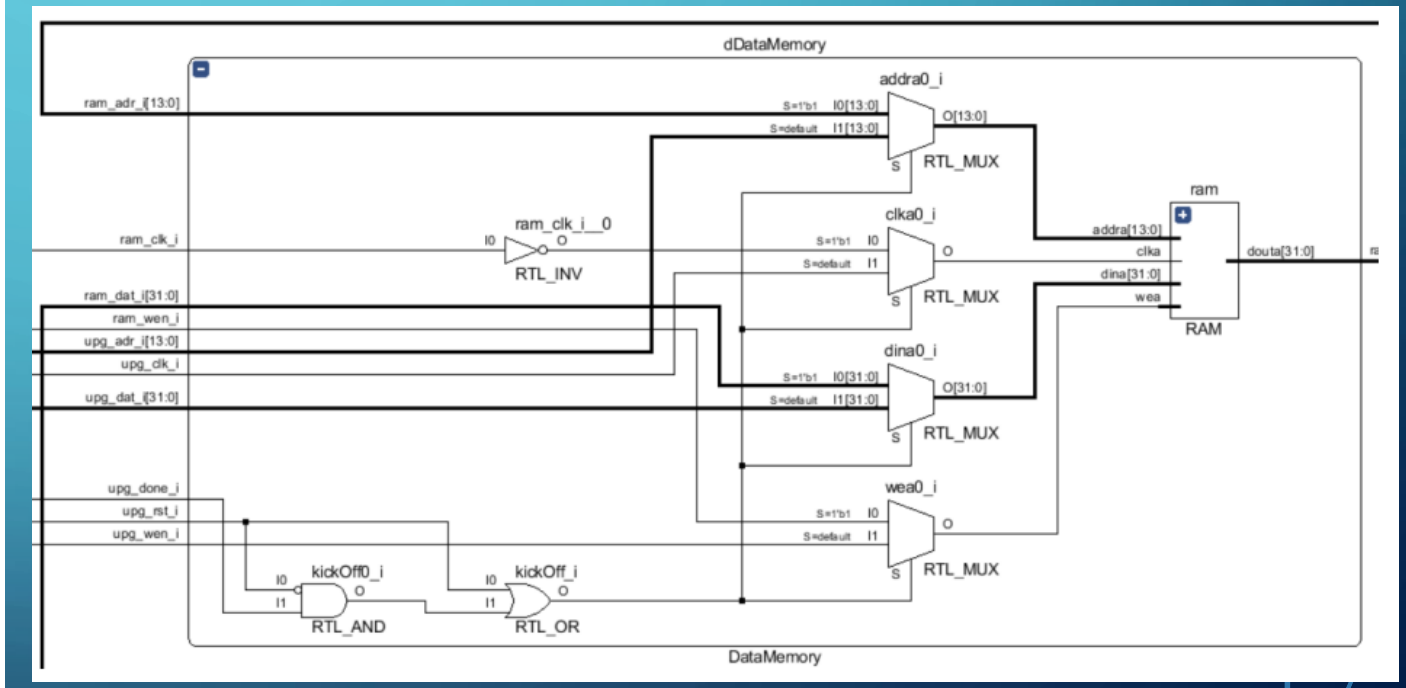
- Do the design in structure design by using the instances of module.

```
module multiplexer_153_2(out1,out2,c10,c11,c12,c13,a1,b1,g1n,
                c20,c21,c22,c23,a2,b2,g2n);

input c10,c11,c12,c13,a1,b1,g1n,c20,c21,c22,c23,a2,b2,g2n;
output out1,out2;

multiplexer_153 m1(
                .g1n(g1n),
                .a(a1),
                .b(b1),
                .c0(c10),
                .c1(c11),
                .c2(c12),
                .c3(c13),
                .out(out1)
                );

multiplexer_153 m2(
                .g1n(g2n),
                .a(a2),
                .b(b2),
                .c0(c20),
                .c1(c21),
                .c2(c22),
                .c3(c23),
                .out(out2)
);

endmodule
```

STRUCTURE DESIGN IN VERILOG(3)

- The demo of sturcture design about CPU.





PRACTICES(2)

Implement the digtial logic circuit and test its function: F(x,y,z) = x^y^z

- Using structure design style in verilog to implement the circuit design in Sum-of-Minterms and Product-of-Maxterms respectively.
- Write the testbench in Verilog to verify the function of the design
- Generate the bitstream and program the device to test the function