2

АЛУ. Верификация

Микропроцессорные средства и системы

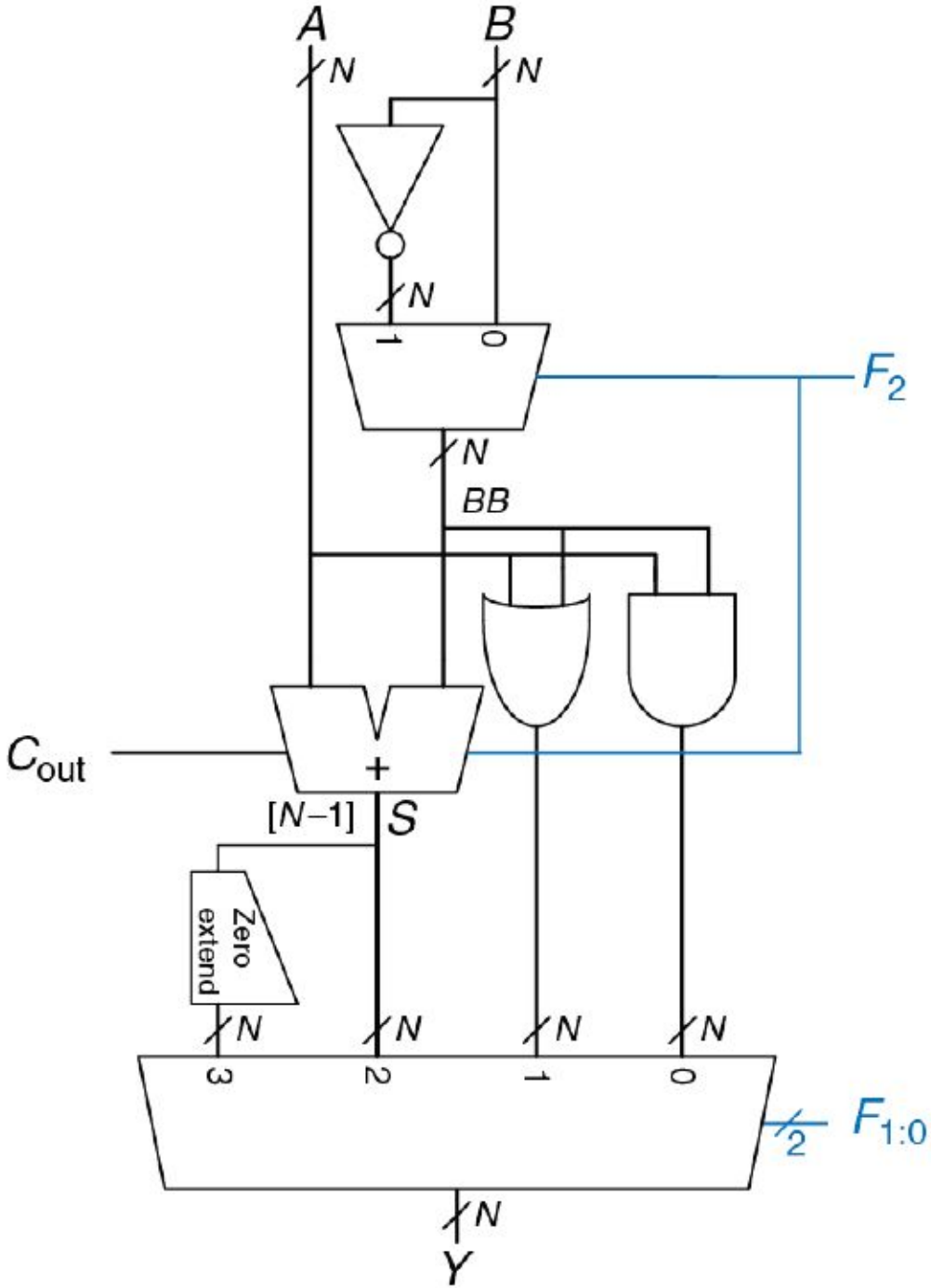
План лабораторной работы

- Арифметико-логическое устройство (**T**)
- Описание АЛУ на Verilog HDL (**S**)
- Верификация АЛУ (**S**)
- Проверка на отладочном стенде (**S**)

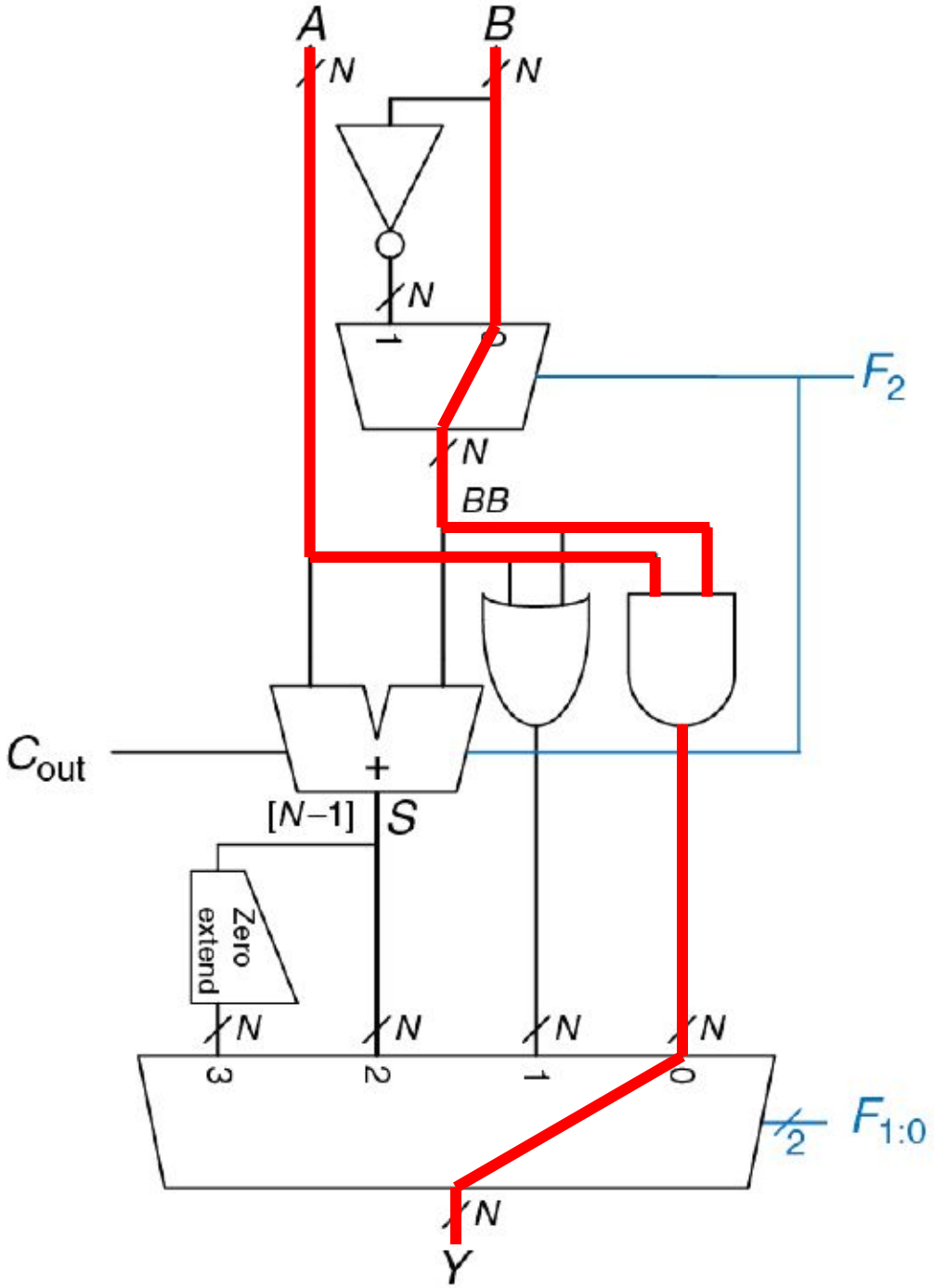
Арифметико-логическое устройство

- АЛУ – блок процессора, выполняющий арифметические и поразрядно логические операции
  - Арифметические операции имеют перенос
  - Логические операции без переноса
- АЛУ – комбинационная схема
- На вход АЛУ поступают информационные сигналы (данные, над которыми происходит операция) и управляющие сигналы (определяют, какая операция будет произведена над данными), на выходе – результат операции и флаги

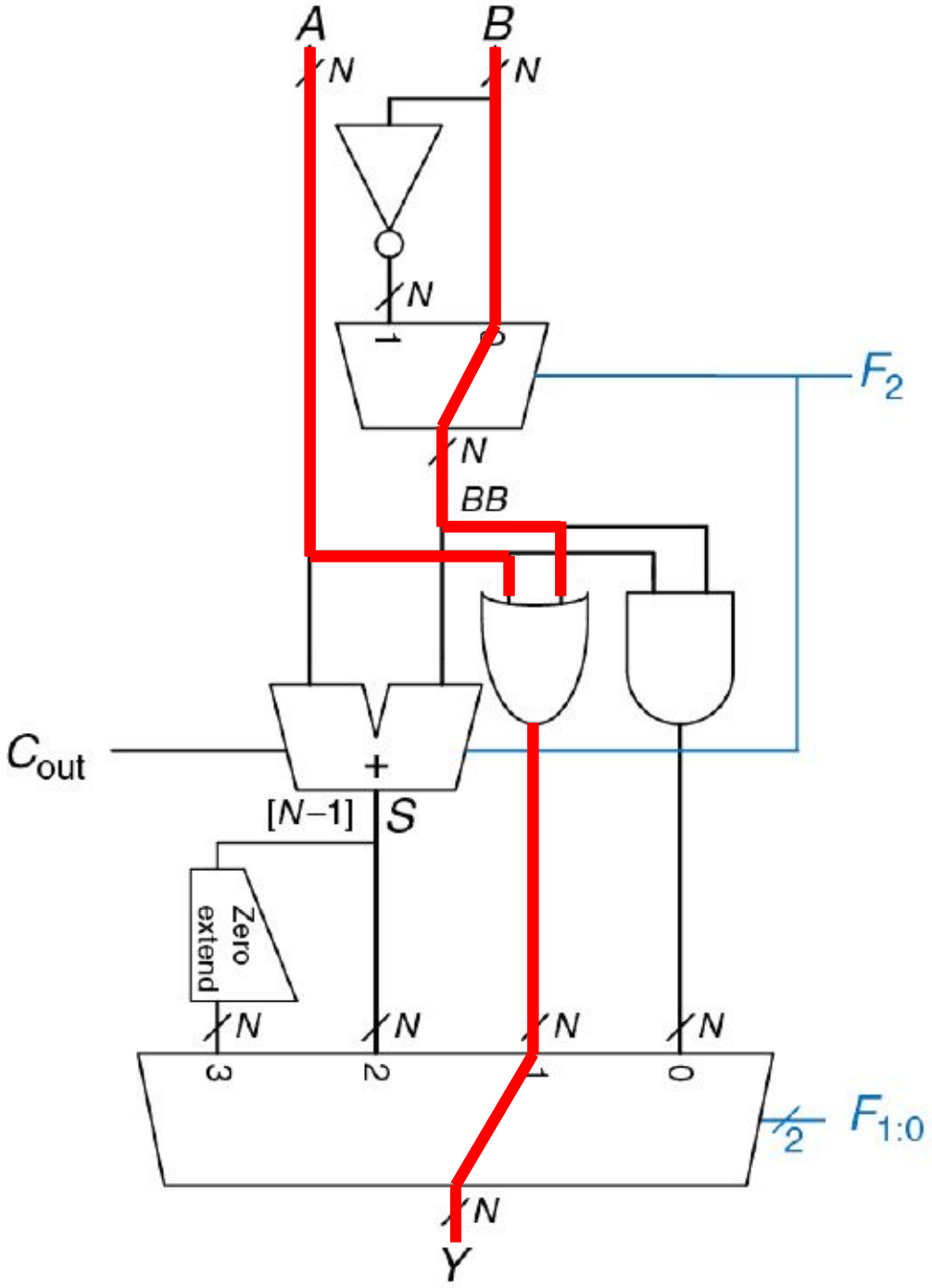
Пример АЛУ



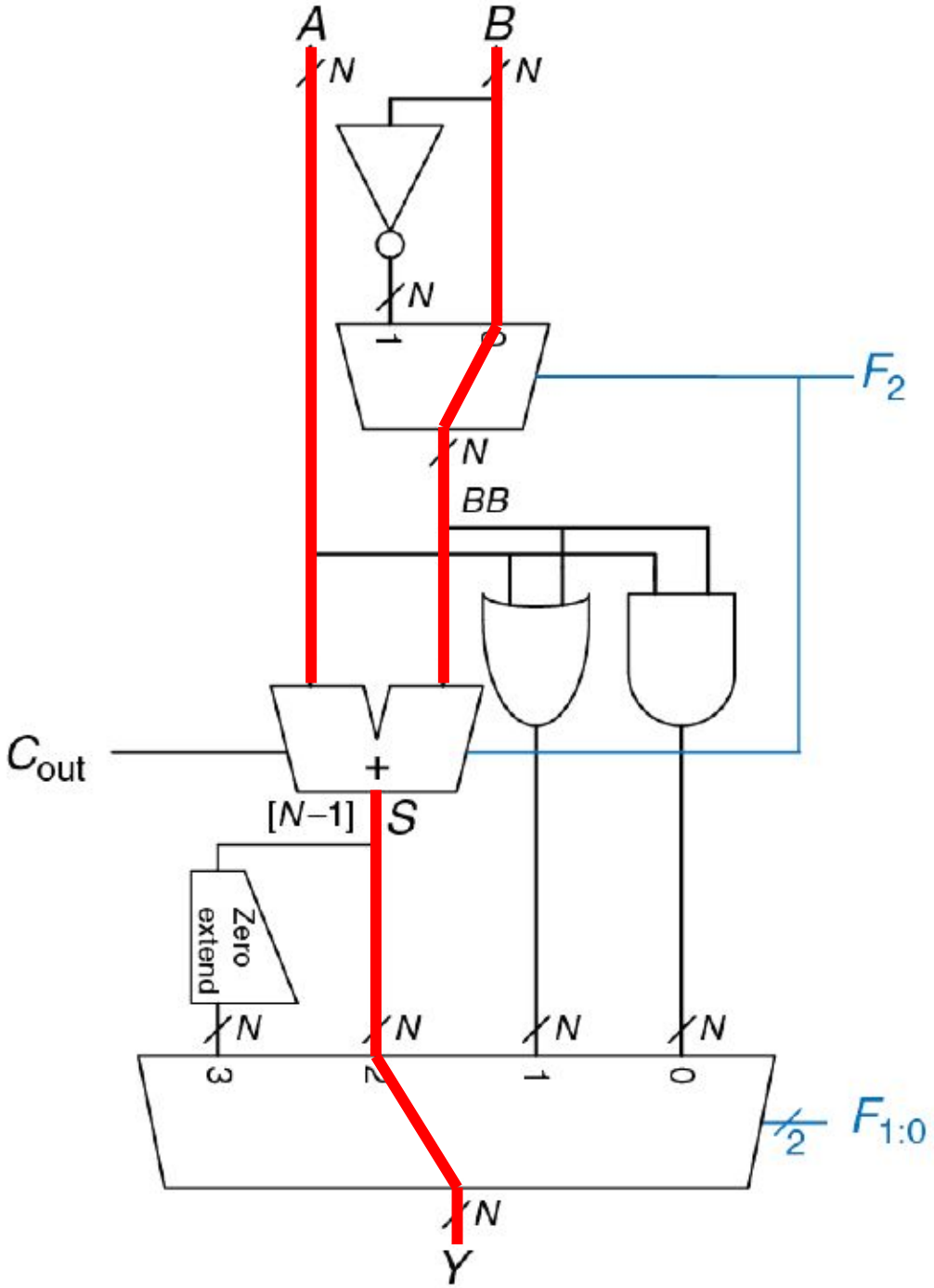
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



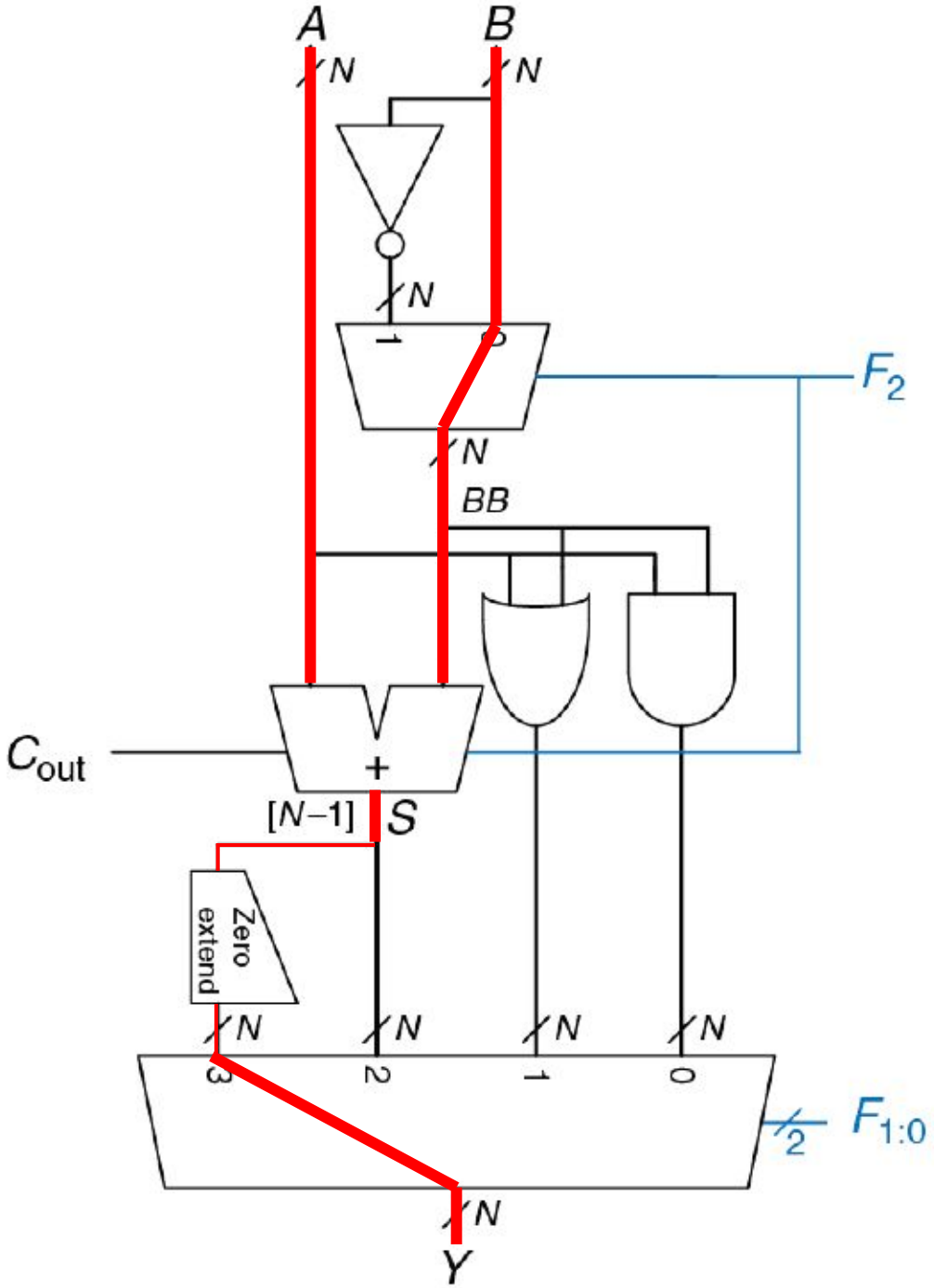
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



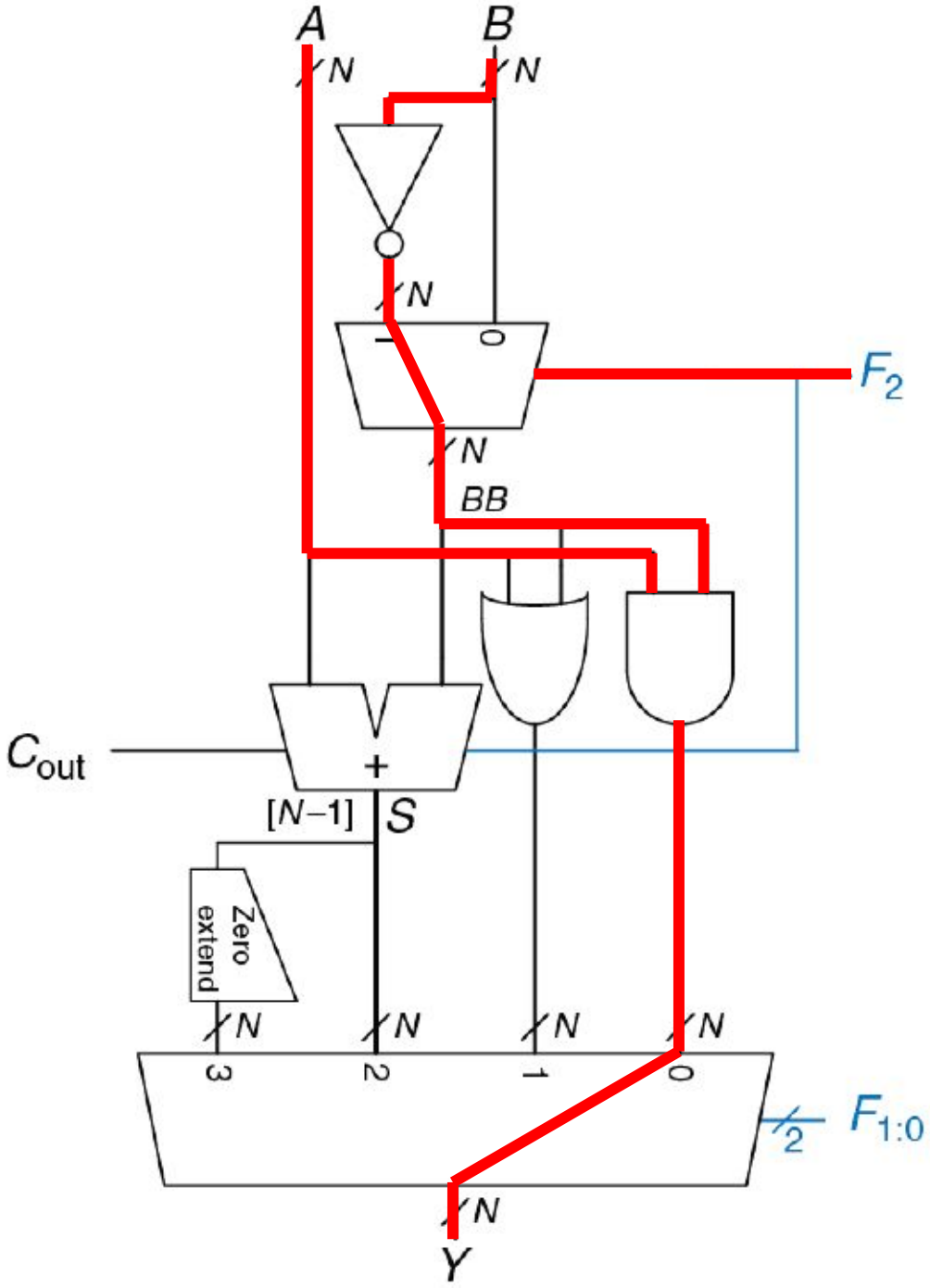
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



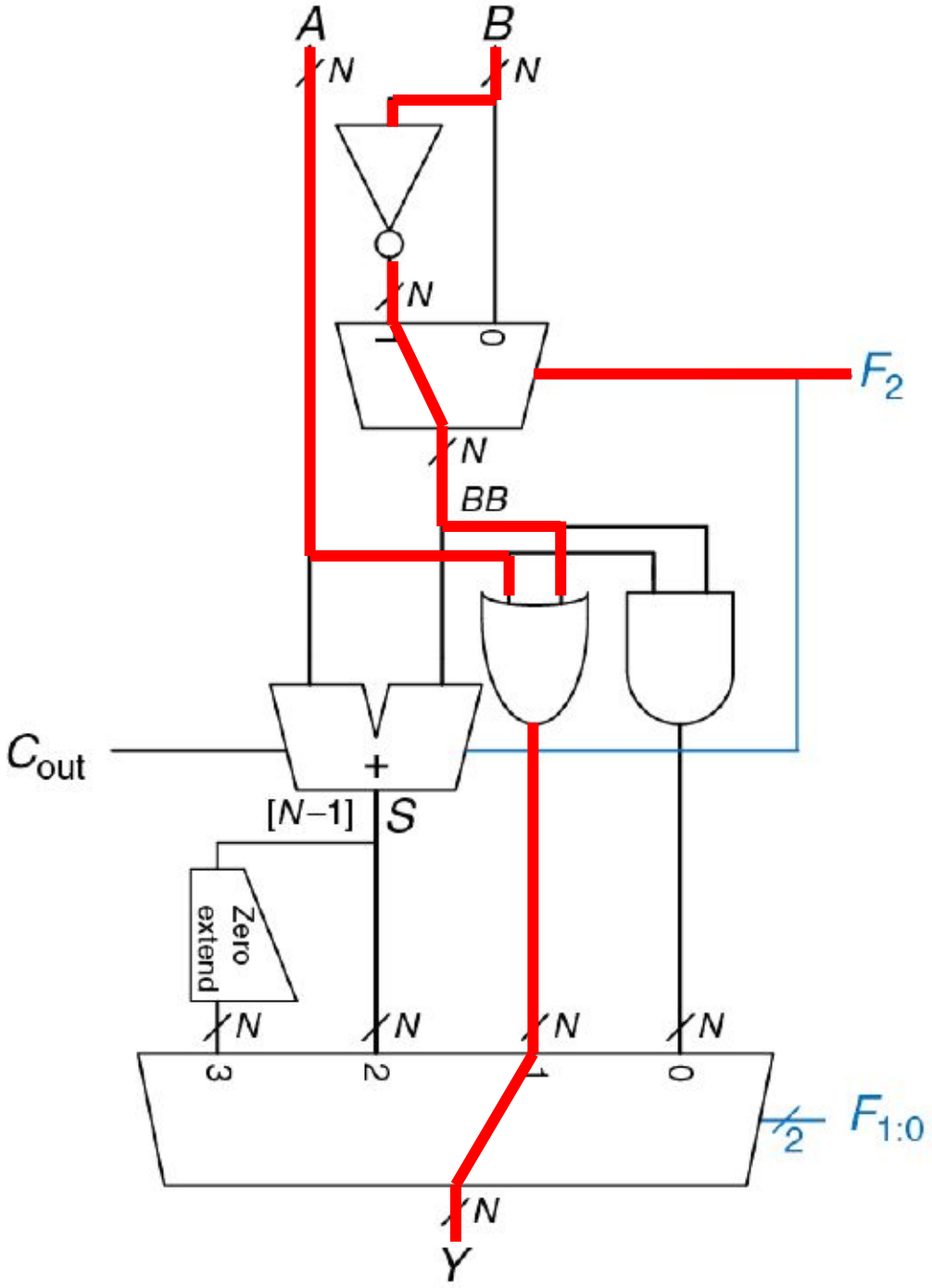
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



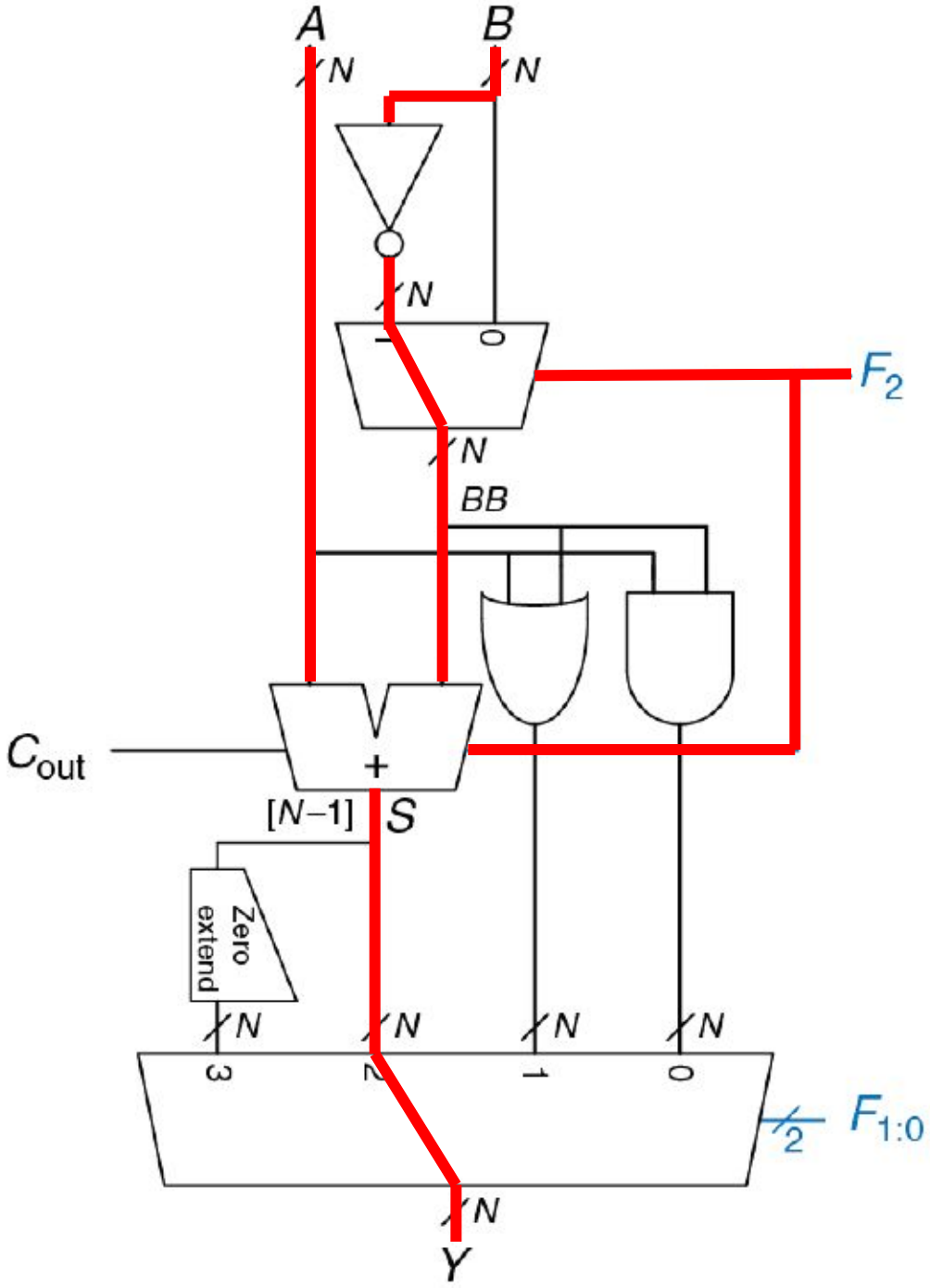
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



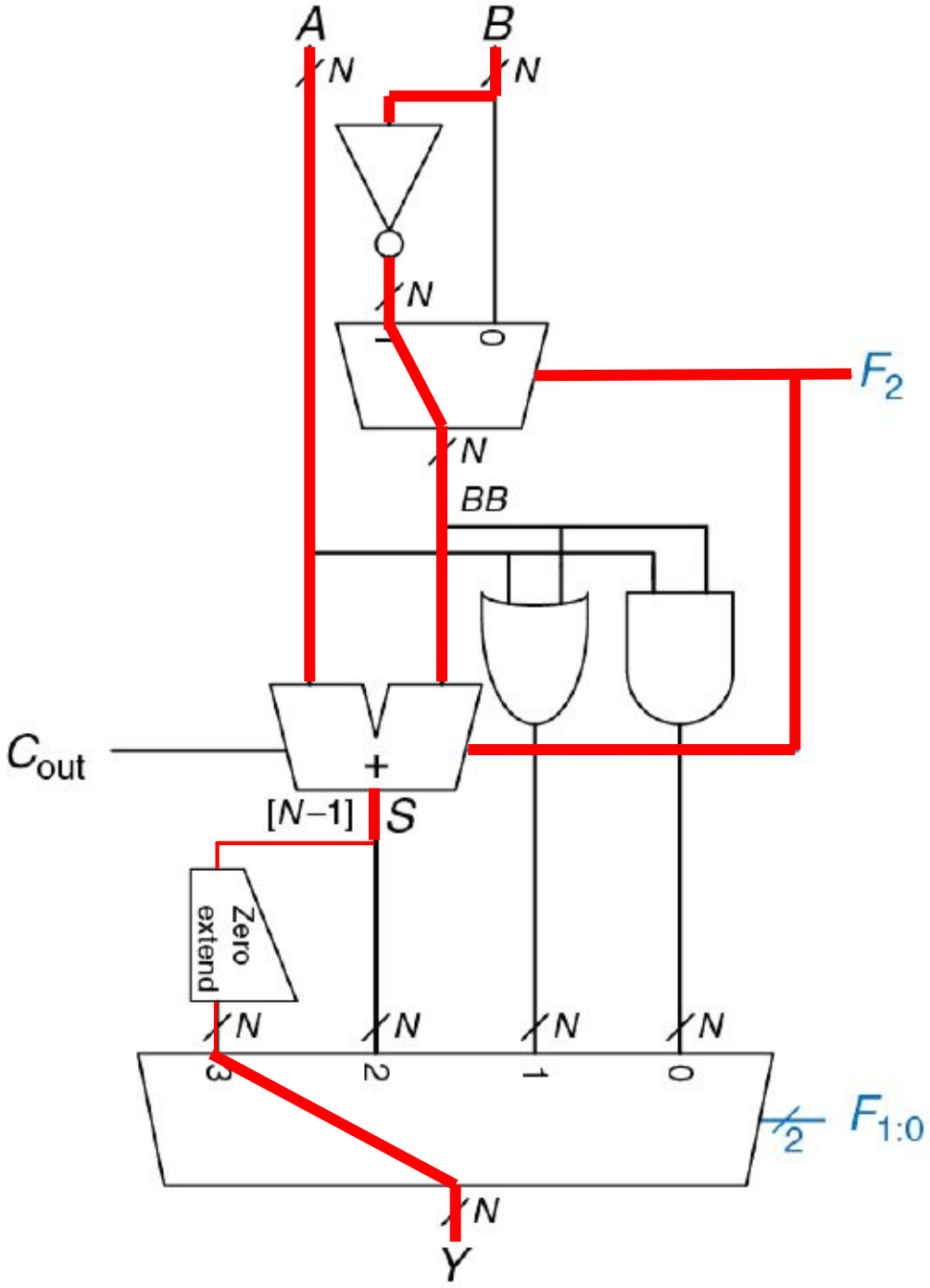
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



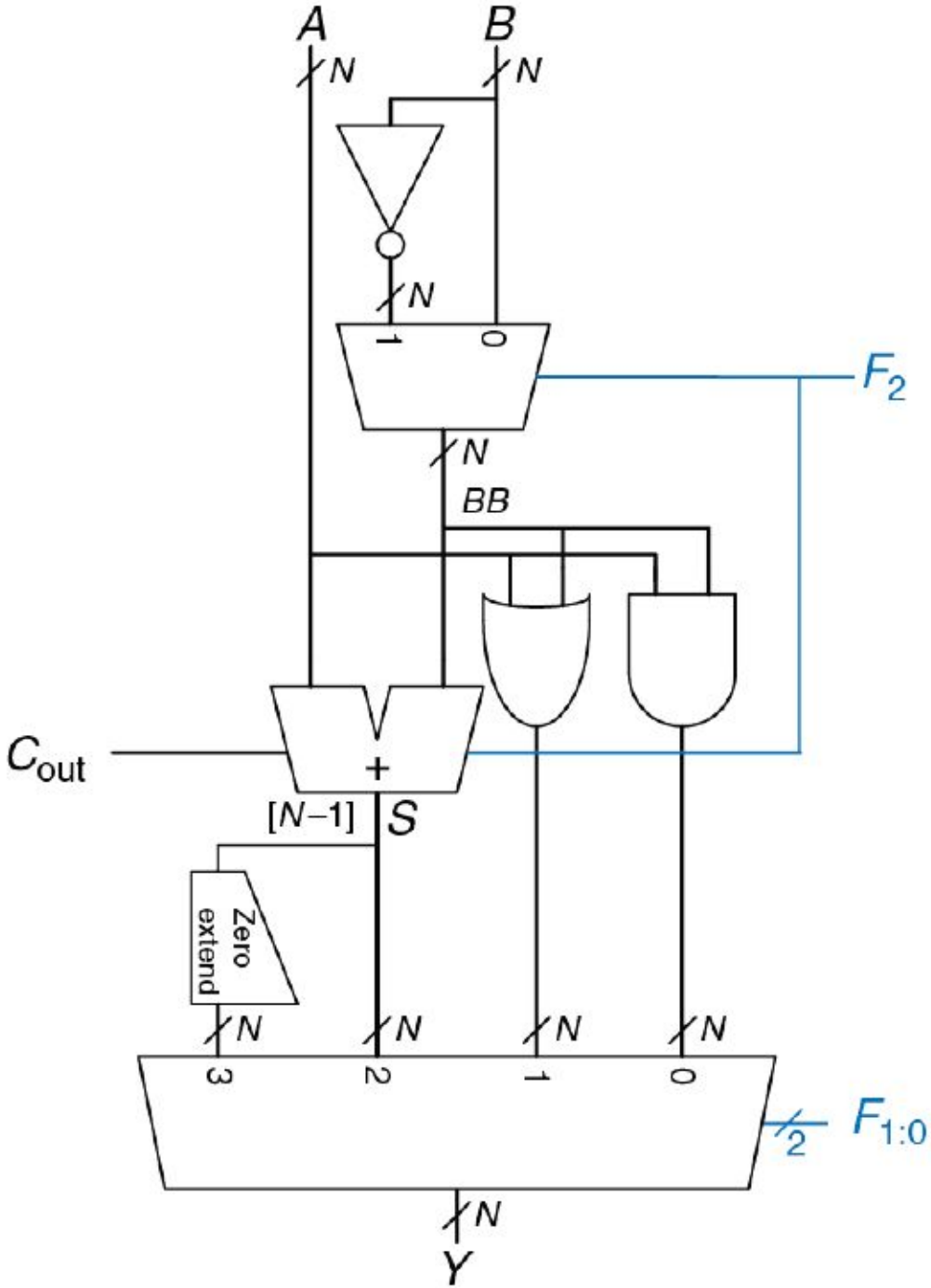
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

План лабораторной работы

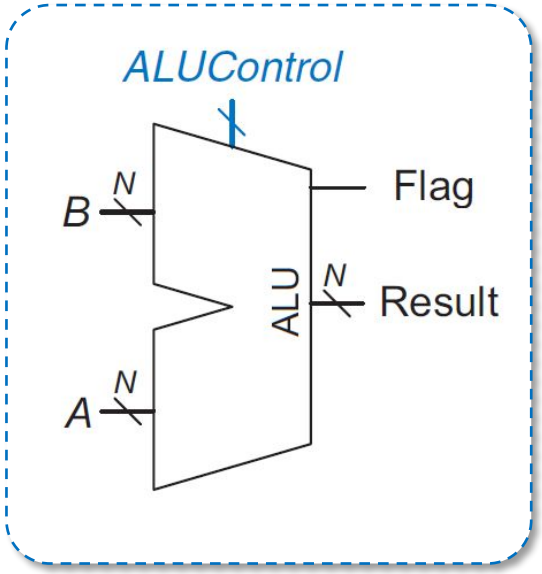
- ~~Арифметико-логическое устройство (**T**)~~
- Описание АЛУ на Verilog HDL (**S**)
- Верификация АЛУ (**S**)
- Проверка на отладочном стенде (**S**)

АЛУ RISC-V

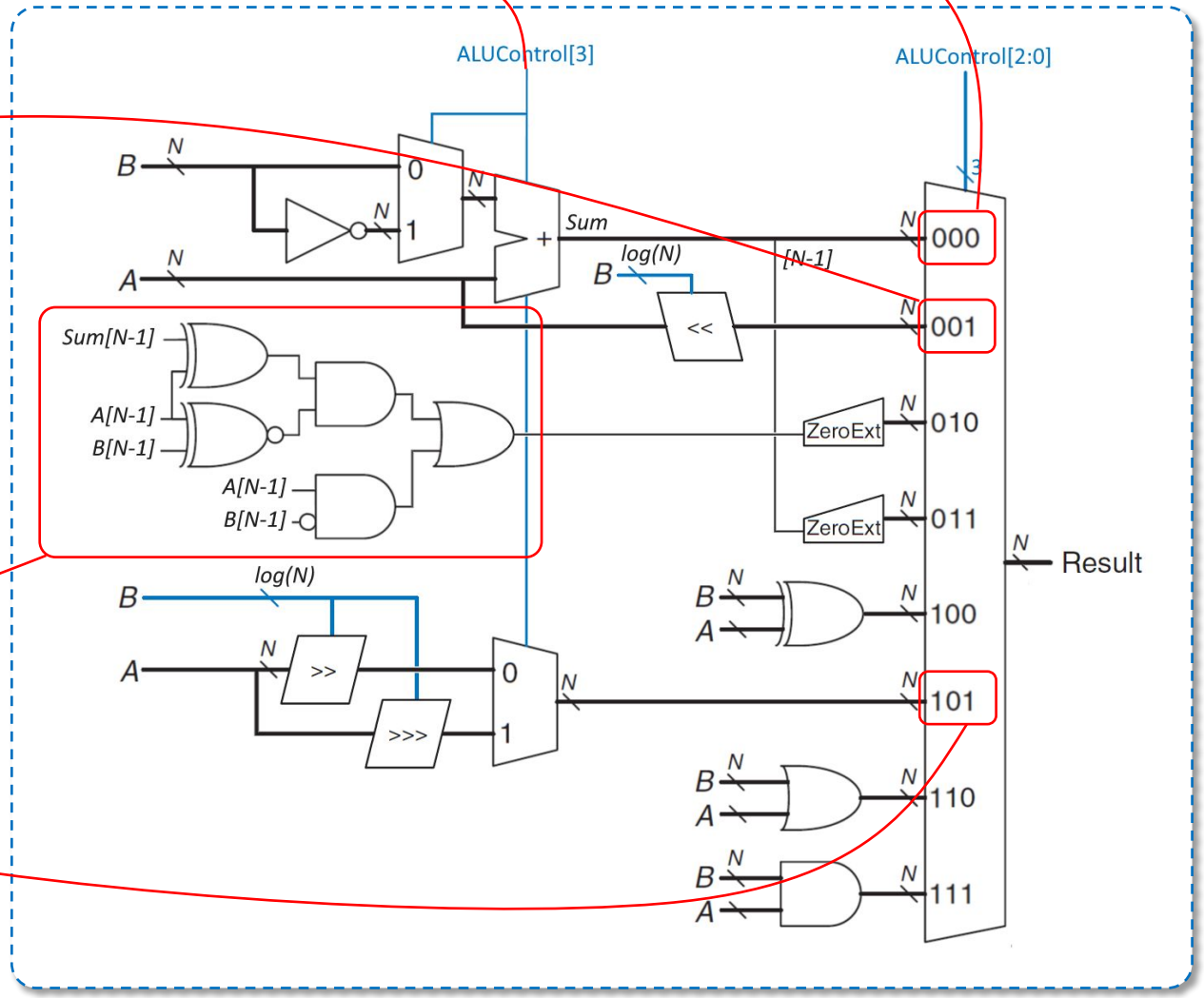
```
`define ADD 5'b00000
…
…
case (ALUOp)
        `ADD : Result = …
…
…
```

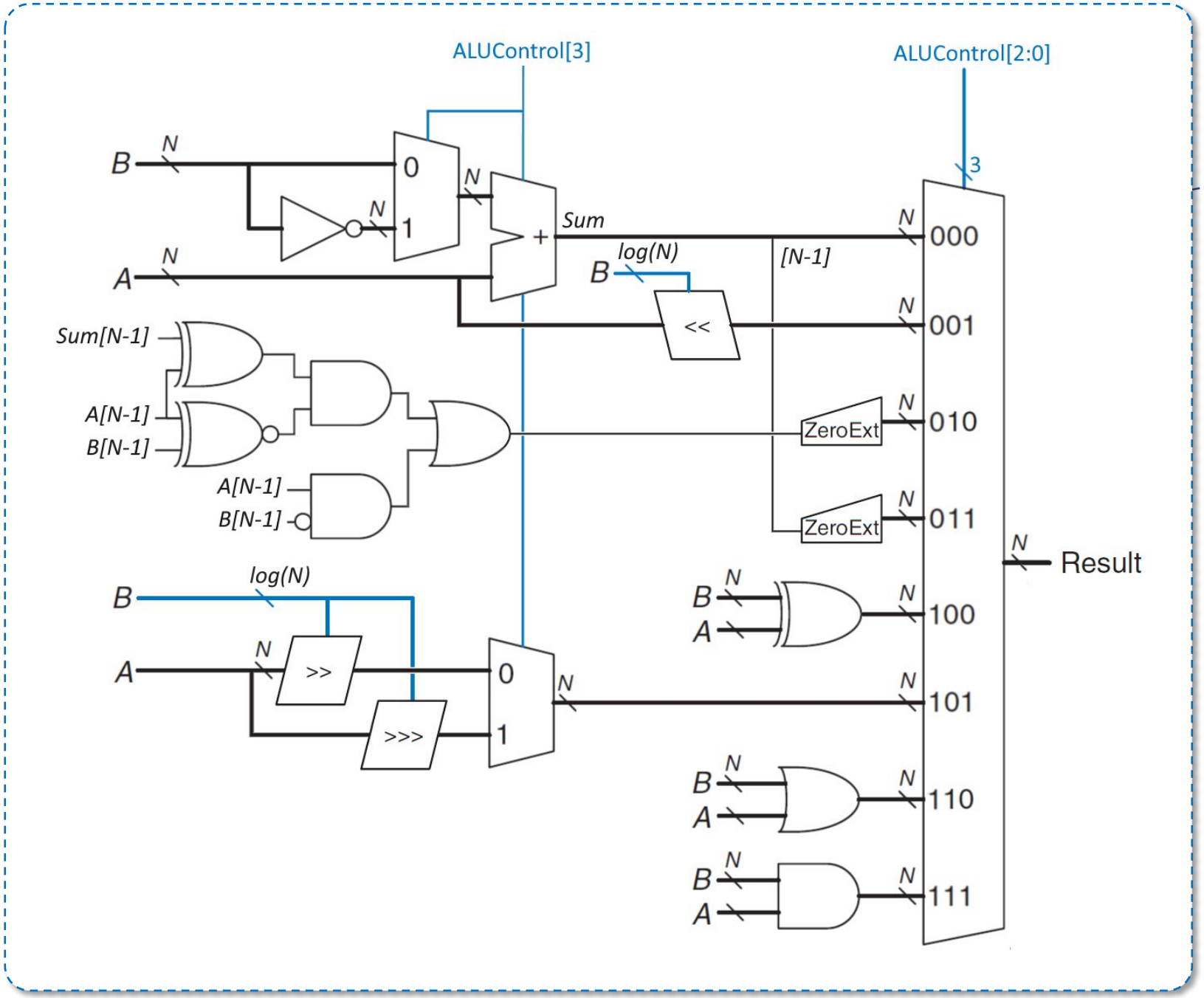
ALUOp = {flag, addsub, aluop}		
0 0 000	Result = A + B	Flag = 0
0 1 000	Result = A - B	Flag = 0
0 0 001	Result = A << B	Flag = 0
0 0 010	Result = signed(A < B)	Flag = 0
0 0 011	Result = (A < B)	Flag = 0
0 0 100	Result = A ^ B	Flag = 0
0 0 101	Result = A >> B	Flag = 0
0 1 101	Result = signed(A) >>> B	Flag = 0
0 0 110	Result = A | B	Flag = 0
0 0 111	Result = A & B	Flag = 0
1 1 000	Result = 0	Flag = (A == B)
1 1 001	Result = 0	Flag = (A != B)
1 1 100	Result = 0	Flag = signed(A < B)
1 1 101	Result = 0	Flag = signed(A ≥ B)
1 1 110	Result = 0	Flag = (A < B)
1 1 111	Result = 0	Flag = (A > B)

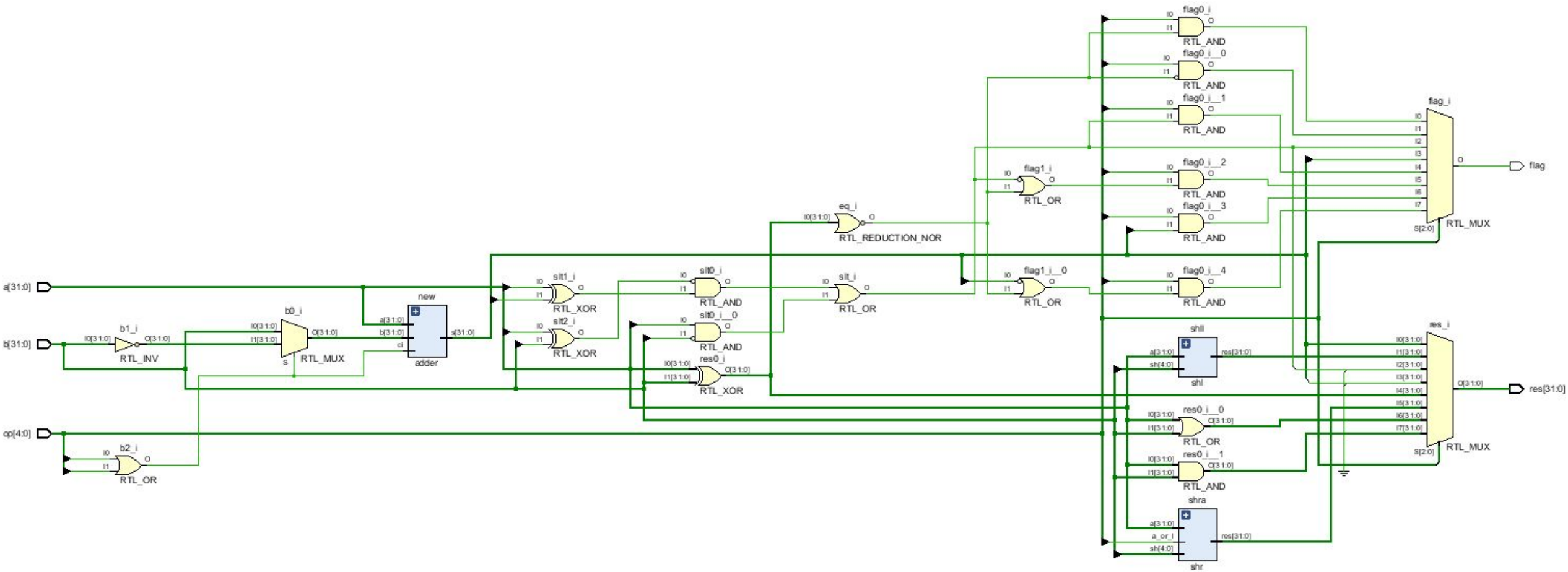
```
ule ALU_RISCV (
    ut  [4:0]     ALUOp,
    ut  [31:0]    A,
  input  [31:0]   B,
  output [31:0]   Result,
  output          Flag
);
```



funct3	funct7	Description	Operation
000	0000000	add	rd = rs1 + rs2
000	0100000	sub	rd = rs1 − rs2
001	0000000	shift left logical	rd = rs1 << $rs2_{4:0}$
010	0000000	set less than	rd = (rs1 < rs2)
011	0000000	set less than unsigned	rd = (rs1 < rs2)
100	0000000	xor	rd = rs1 ^ rs2
101	0000000	shift right logical	rd = rs1 >> $rs2_{4:0}$
101	0100000	shift right arithmetic	rd = rs1 >>> $rs2_{4:0}$
110	0000000	or	rd = rs1 | rs2
111	0000000	and	rd = rs1 & rs2
000	–	branch if =	if (rs1 == rs2) PC = BTA
001	–	branch if ≠	if (rs1 ≠ rs2) PC = BTA
100	–	branch if <	if (rs1 < rs2) PC = BTA
101	–	branch if ≥	if (rs1 ≥ rs2) PC = BTA
110	–	branch if < unsigned	if (rs1 < rs2) PC = BTA
111	–	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA

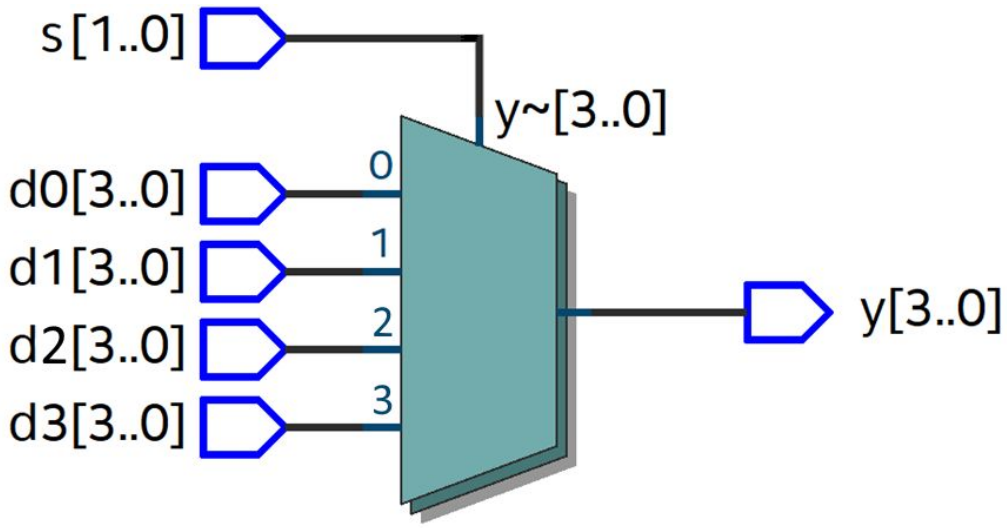



ALUControl[3]
ALUControl[2:0]
B
A
N
0
1
+
Sum
log(N)
[N-1]
<<
3
000
001
010
011
100
101
110
111
Result
Sum[N-1]
A[N-1]
B[N-1]
ZeroExt
>>
>>>
ALUControl
Flag
ALU
Result


a[31:0]
b[31:0]
op[4:0]
b1_i
I0[31:0]
O[31:0]
RTL_INV
b0_i
I0[31:0]
I1[31:0]
O[31:0]
S
RTL_MUX
new
a[31:0]
b[31:0]
ci
s[31:0]
adder
b2_i
I0
I1
O
RTL_OR
slt1_i
RTL_XOR
slt2_i
RTL_XOR
slt0_i
RTL_AND
slt0_i__0
RTL_AND
res0_i
I0[31:0]
I1[31:0]
O[31:0]
RTL_XOR
eq_i
I0[31:0]
RTL_REDUCTION_NOR
slt_i
RTL_OR
flag1_i
RTL_OR
flag1_i__0
RTL_OR
flag0_i
RTL_AND
flag0_i__0
RTL_AND
flag0_i__1
RTL_AND
flag0_i__2
RTL_AND
flag0_i__3
RTL_AND
flag0_i__4
RTL_AND
flag_i
I0
I1
I2
I3
I4
I5
I6
I7
S[2:0]
RTL_MUX
flag
shll
a[31:0]
sh[4:0]
res[31:0]
shl
res0_i__0
I0[31:0]
I1[31:0]
O[31:0]
RTL_OR
res0_i__1
RTL_AND
shra
a[31:0]
a_or_l
sh[4:0]
res[31:0]
shr
res_i
I0[31:0]
I1[31:0]
I2[31:0]
I3[31:0]
I4[31:0]
I5[31:0]
I6[31:0]
I7[31:0]
S[2:0]
O[31:0]
RTL_MUX
res[31:0]

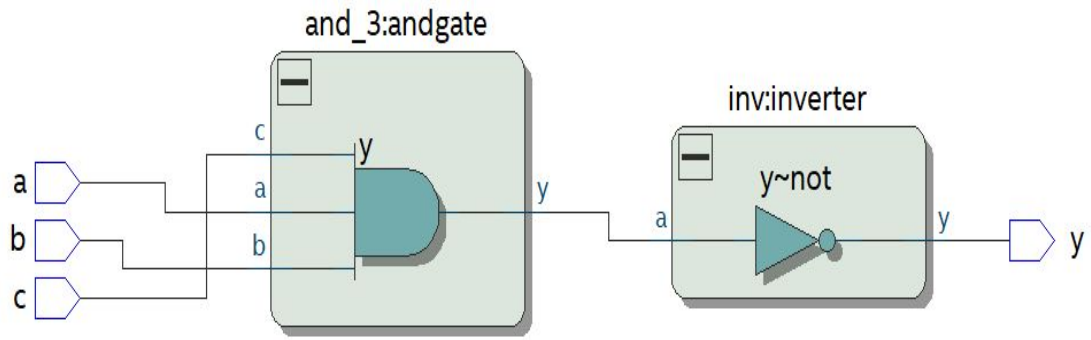
CASE

```
module mux (
        input           [3:0]  d0, d1, d2, d3,
        input           [1:0]  s,
        output      reg [3:0]  y
);

always @ (*) begin
        case (s)
                2'b00:      y = d0;
                2'b01:      y = d1;
                2'b10:      y = d2;
                2'b11:      y = d3;
        endcase
end

endmodule
```



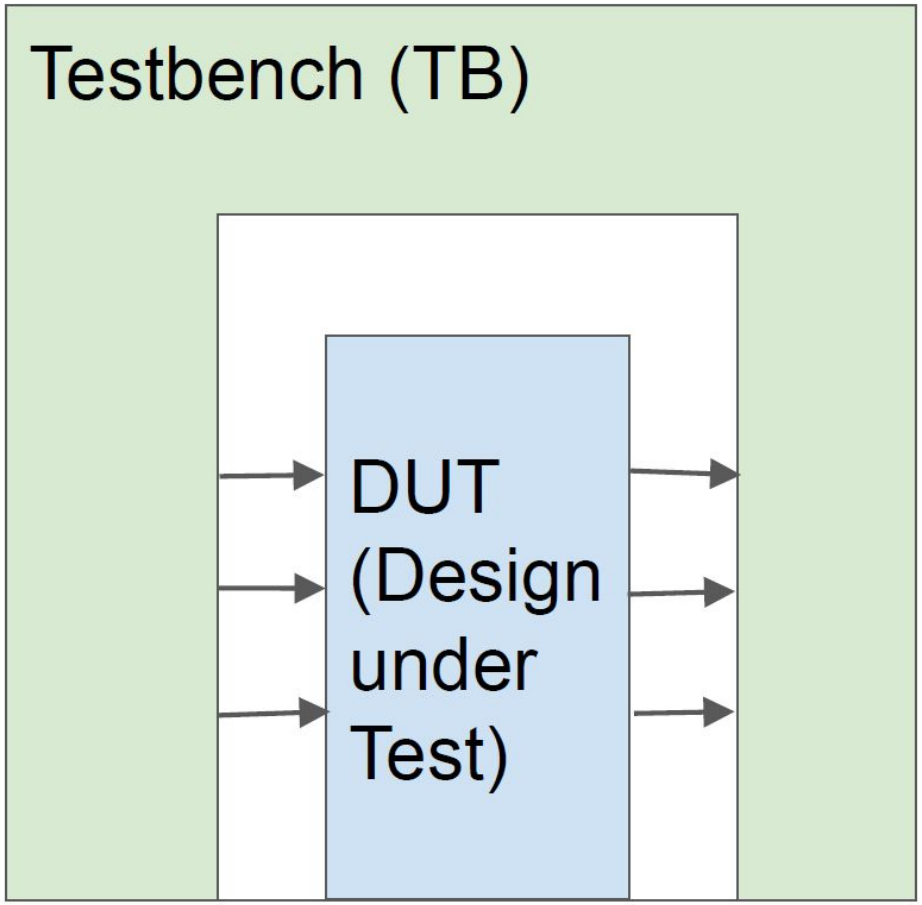
Иерархия модулей в Verilog

```
module dut (
        input       a, b, c,
        output      y
);

wire n1;

and_3 andgate (
        .a(a),
        .b(b),
        .c(c),
        .y(n1)
);

inv inverter (
        .a(n1),
        .y(y)
);

endmodule
```

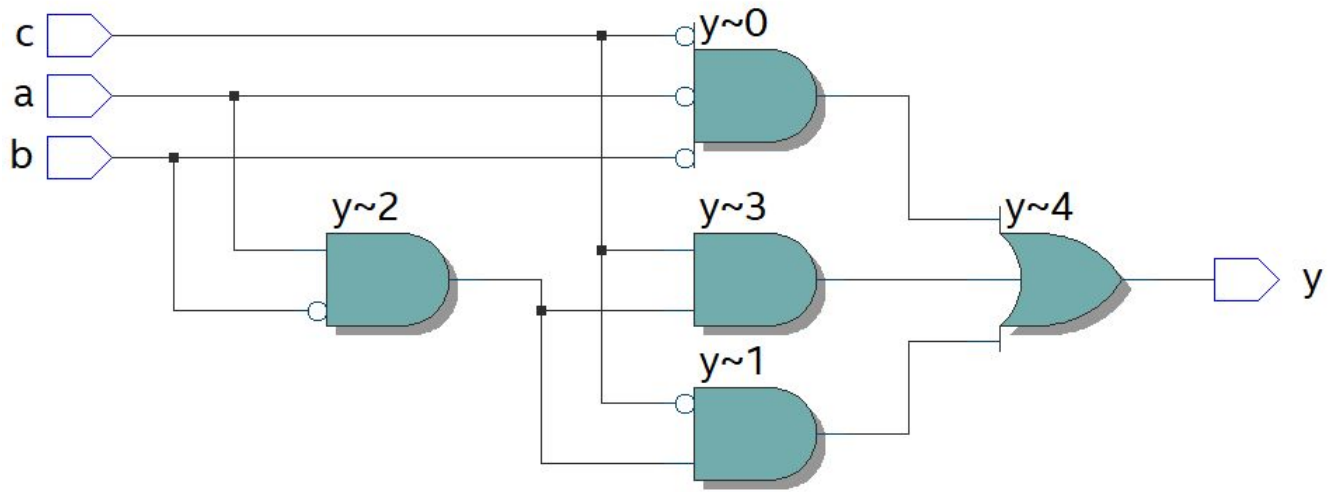


- Имя подключаемого модуля (**and_3**, **inv**)
- Название примитива. Например, нам может понадобиться **3** копии модуля **and_3**. Тогда мы сможем подключить 3 экземпляра модуля **and_3**, используя различные наименования для прототипов (andgate_1, andgate_2 …)
- Символ точка, перед наименованием порта отсылает к реальному порту подключаемого модуля (у модуля **inverter**, порты именуются **a, y**). В скобках обозначается куда будут подключаться сигналы в *top*-модуле

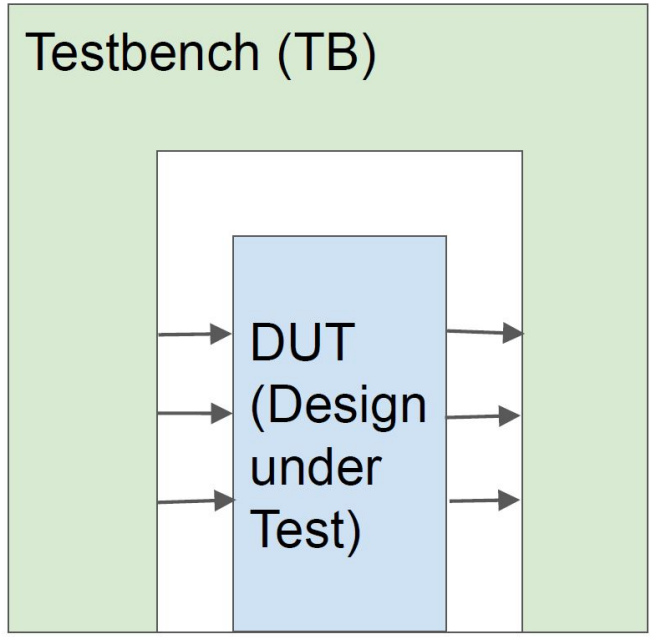
Тестовое окружение



```
module my_module (
    input   a, b, c,
    output  y
);

assign y = a & ~b & ~c | a & ~b & c | a & ~b & c;

endmodule
```



```
`timescale 1ns / 1ps

module testbench ();

reg  a, b, c;
wire       y;

my_module dut (a, b, c, y);

initial begin
      a = 0; b = 0; c = 0; #10;
      if (y === 1)
            $display("Good");
      else
            $display("Bad");
      c = 1; #10;
end

endmodule
```



```
initial begin
    go_op(6,  3);
     go_op(2, 6);
     go_op(5, 3);
     go_op(8, 2);
     $stop;
end


task go_op;
     input [3:0] a_op, b_op;
  begin
     a = a_op;
     b = b_op;
     #100;
     if (res == (a + b))
          $display("good %d + %d = %d", a, b, LEDR[4:0]);
     else
          $display("bad %d + %d = %d", a, b, LEDR[4:0]);
     #10;
  end
endtask
```

Задание

- Разработать 32-битное АЛУ для архитектуры RISC-V на основе ранее разработанного сумматора
- Написать testbench для верификации разработанного АЛУ
- (если останется время) Проверить работу на стенде