1. Design and implement a full adder. (s+cout=x+y+cin)

From the specifications, determine the inputs, outputs, and their symbols.

Inputs:	x,	(augend)
	y,	(addend)
	cin	(carry in)
Outputs:	s,	(sum)
	cout	(carry out)

Derive the truth table (functions) from the relationship between the inputs and outputs.

x	y	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

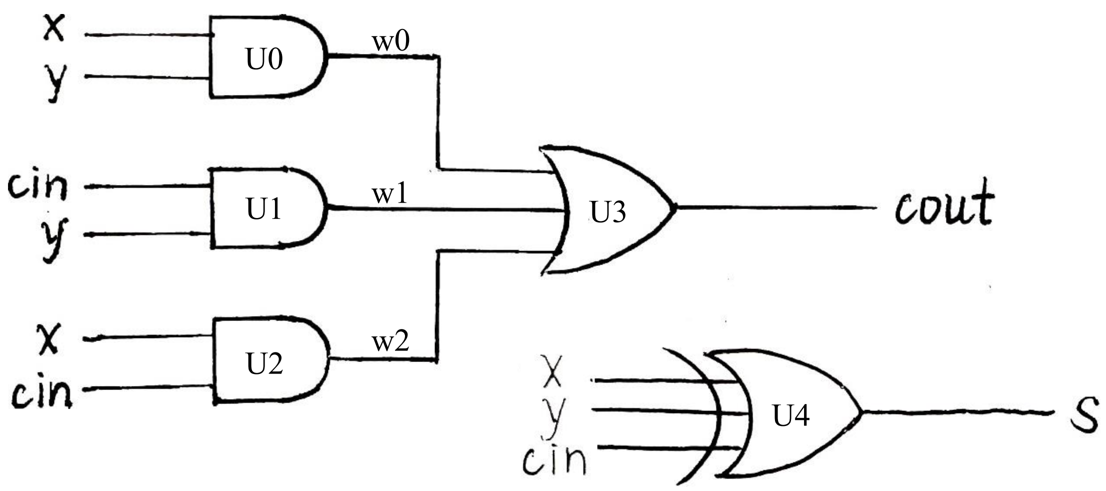
1.1 Write the logic equation.

Derive the simplified Boolean functions for each output function.

$s = x \oplus y \oplus cin$

$cout = x \times y + y \times cin + cin \times x$

1.2 Draw the related logic diagram.



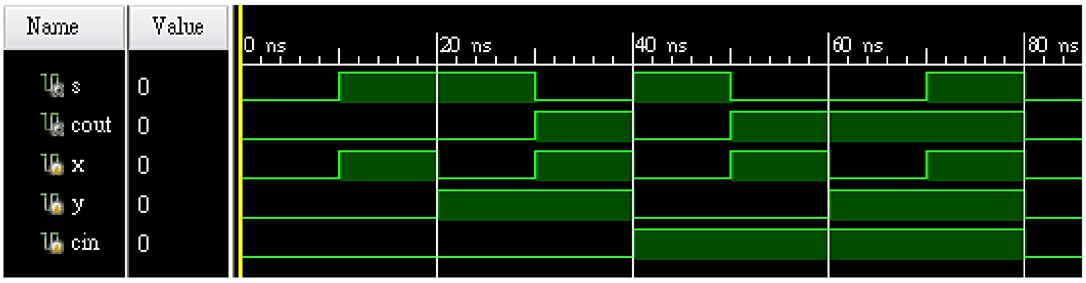
1.3 Verilog RTL representation with verification.

Construct the Verilog code according to the logic diagram.

```
module fulladder(
    output  s,       // sum
    output  cout,    // carry out
    input   x,       // augend
    input   y,       // addend
    input   cin      // carry in
    );

    assign  cout = x & y | y & cin | cin & x;
    assign  s = x ^ y ^ cin;

endmodule           // fulladder
```

Write the test bench and verify the design.

```
module fulladder_test();
    // outputs
    wire     s;            // sum
    wire     cout;         // carry out

    // inputs
    reg      x;            // augend
    reg      y;            // addend
    reg      cin;          // carry in

    // DUT
    fulladder fa0(
        .s(s),             // sum
        .cout(cout),       // carry out
        .x(x),             // augend
        .y(y),             // addend
        .cin(cin)          // carry in
        );

    // stimulus
    always  #10 {cin, y, x} = {cin, y, x} + 1;
    initial {cin, y, x} = 0;

endmodule                  // fulladder_test
```



2. Design a 3-to-8-line decoder with enable (input in[2:0], enable, output d[7:0]).

From the specifications, determine the inputs, outputs, and their symbols.

Inputs: in[2:0], en

Outputs: d[7:0]

Derive the truth table from the relationship between the inputs and outputs.

en	in[2]	in[1]	in[0]	d[7]	d[6]	d[5]	d[4]	d[3]	d[2]	d[1]	d[0]
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

2.1 Logic equation.

Derive the simplified Boolean functions for each output function.

d[0] = en × in[2]’ × in[1]’ × in[0]’

d[1] = en × in[2]’ × in[1]’ × in[0]

d[2] = en × in[2]’ × in[1] × in[0]’

d[3] = en × in[2]’ × in[1] × in[0]

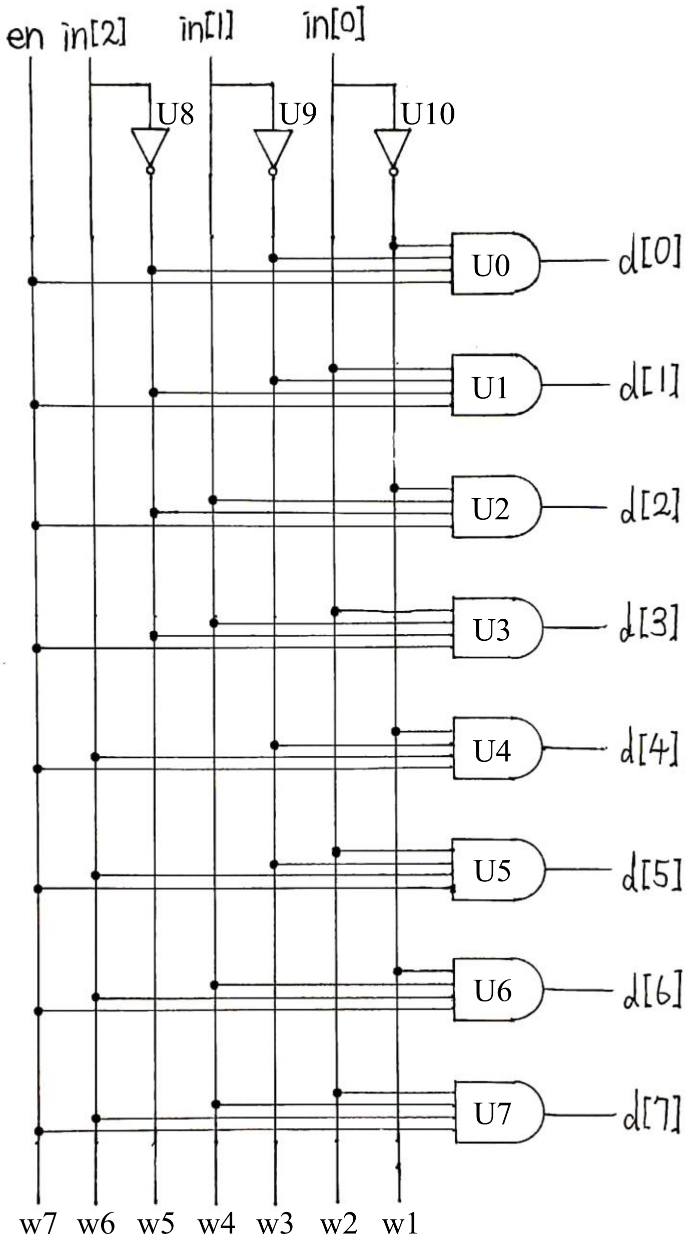
d[4] = en × in[2] × in[1]’ × in[0]’

d[5] = en × in[2] × in[1]’ × in[0]

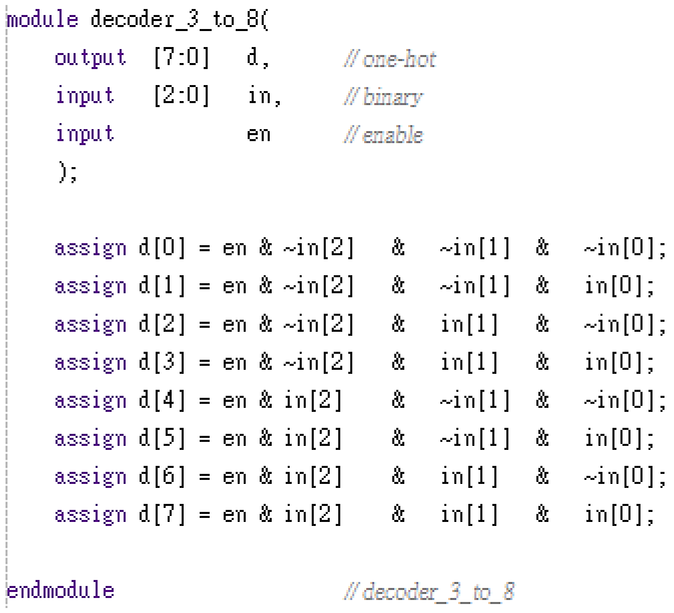
d[6] = en × in[2] × in[1] × in[0]’

d[7] = en × in[2] × in[1] × in[0]

2.2 Logic schematic.



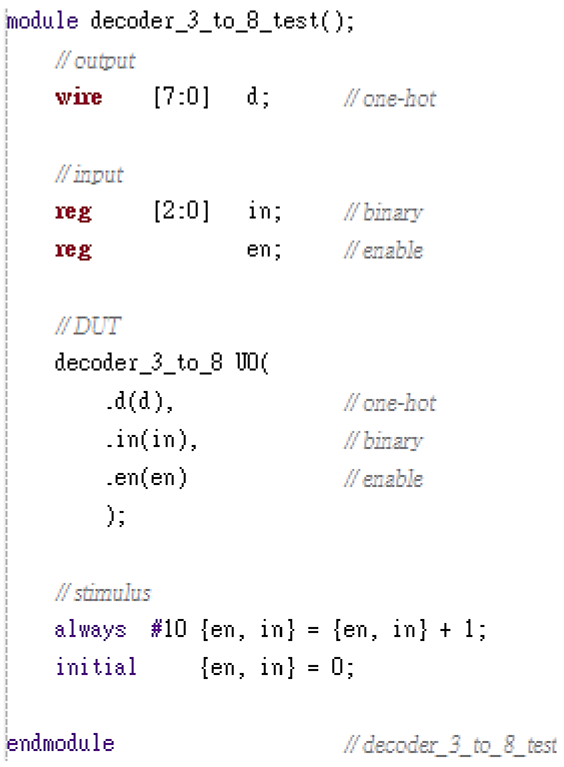
Construct the Verilog code according to the logic diagram.



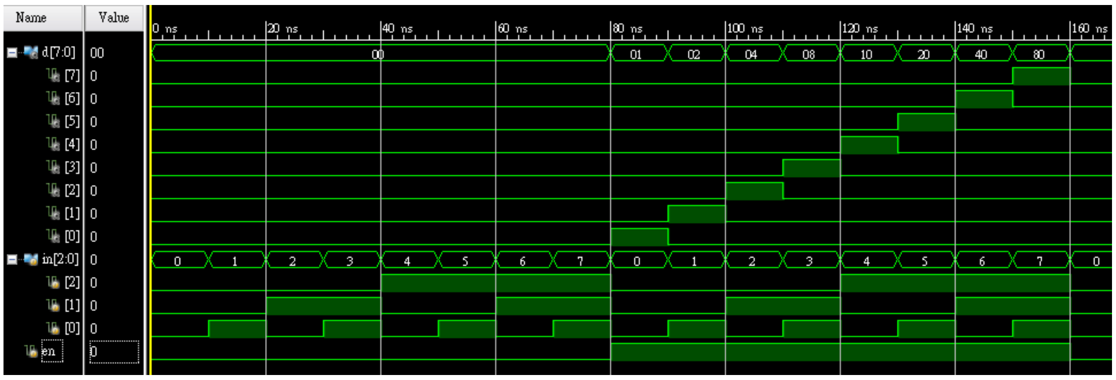
```
module decoder_3_to_8(
    output  [7:0]   d,       // one-hot
    input   [2:0]   in,      // binary
    input           en       // enable
    );

    assign d[0] = en & ~in[2]   &   ~in[1]  &   ~in[0];
    assign d[1] = en & ~in[2]   &   ~in[1]  &   in[0];
    assign d[2] = en & ~in[2]   &   in[1]   &   ~in[0];
    assign d[3] = en & ~in[2]   &   in[1]   &   in[0];
    assign d[4] = en & in[2]    &   ~in[1]  &   ~in[0];
    assign d[5] = en & in[2]    &   ~in[1]  &   in[0];
    assign d[6] = en & in[2]    &   in[1]   &   ~in[0];
    assign d[7] = en & in[2]    &   in[1]   &   in[0];

endmodule                    // decoder_3_to_8
```

2.3 Verilog RTL representation with verification.

Write the test bench and verify the design.



```
module decoder_3_to_8_test();
    // output
    wire    [7:0]   d;       // one-hot

    // input
    reg     [2:0]   in;      // binary
    reg             en;      // enable

    // DUT
    decoder_3_to_8 U0(
        .d(d),               // one-hot
        .in(in),             // binary
        .en(en)              // enable
        );

    // stimulus
    always  #10 {en, in} = {en, in} + 1;
    initial     {en, in} = 0;

endmodule                    // decoder_3_to_8_test
```



3. For two 3-bit unsigned numbers a(a2a1a0) and b(b2b1b0), build a logic circuit to output the larger number.

From the specifications, determine the inputs, outputs, and their symbols.

Inputs:	a[2:0],	(a2a1a0)
	b[2:0]	(b2b1b0)
Outputs:	larger[2:0]	

Derive the truth table (functions) from the relationship between the inputs and outputs.

MUX:

sel	a	b	Q
0	0	×	0
0	1	×	1
1	×	0	0
1	×	1	1

3-bit comparator:

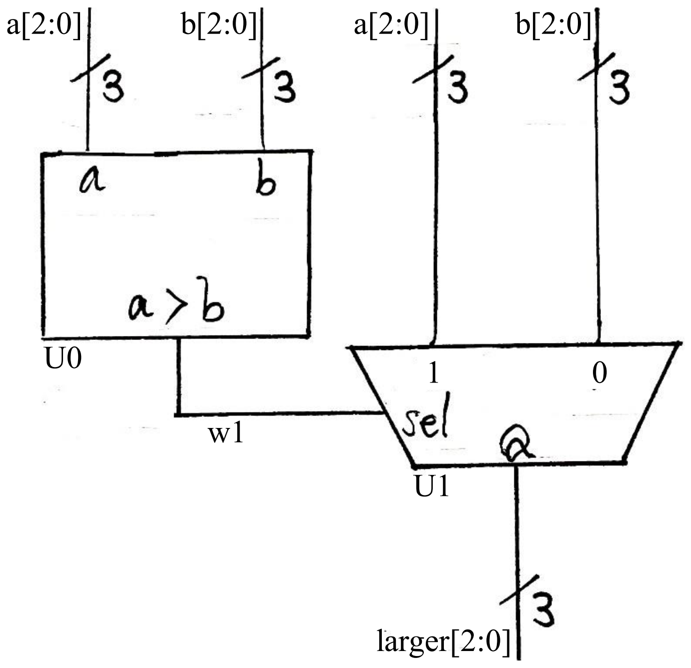
a[2]	a[1]	a[0]	b[2]	b[1]	b[0]	a > b	a = b	a < b
1	×	×	0	×	×	1	0	0
0	1	×	0	0	×	1	0	0
1	1	×	1	0	×	1	0	0
0	0	1	0	0	0	1	0	0
0	1	1	0	1	0	1	0	0
1	0	1	1	0	0	1	0	0
1	1	1	1	1	0	1	0	0
0	×	×	1	×	×	0	0	1
0	0	×	0	1	×	0	0	1

1	0	×	1	1	×	0	0	1
0	0	0	0	0	1	0	0	1
0	1	0	0	1	1	0	0	1
1	0	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	1
0	0	0	0	0	0	0	1	0
0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	1	0
1	0	0	1	0	0	0	1	0
1	0	1	1	0	1	0	1	0
1	1	0	1	1	0	0	1	0
1	1	1	1	1	1	0	1	0

Derive the simplified Boolean functions for each output function.

MUX: $Q = sel \times a + sel' \times b$

comparator: $(a > b) = (a[2] \times b[2]') +$
$(a[1] \times b[1]')(a[2] \odot b[2]') +$
$(a[0] \times b[0]')(a[1] \odot b[1]')(a[2] \odot b[2]')$

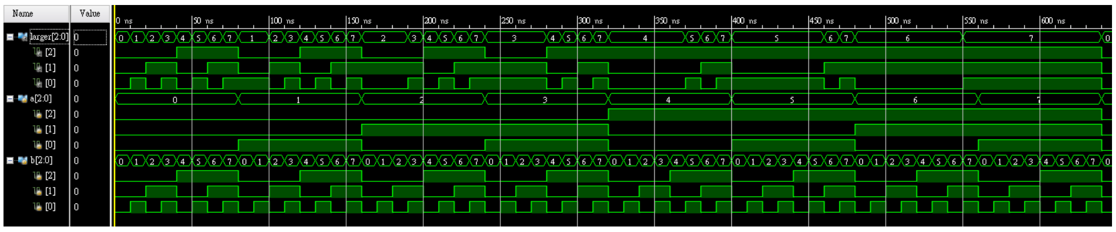


Construct the Verilog code according to the logic diagram.

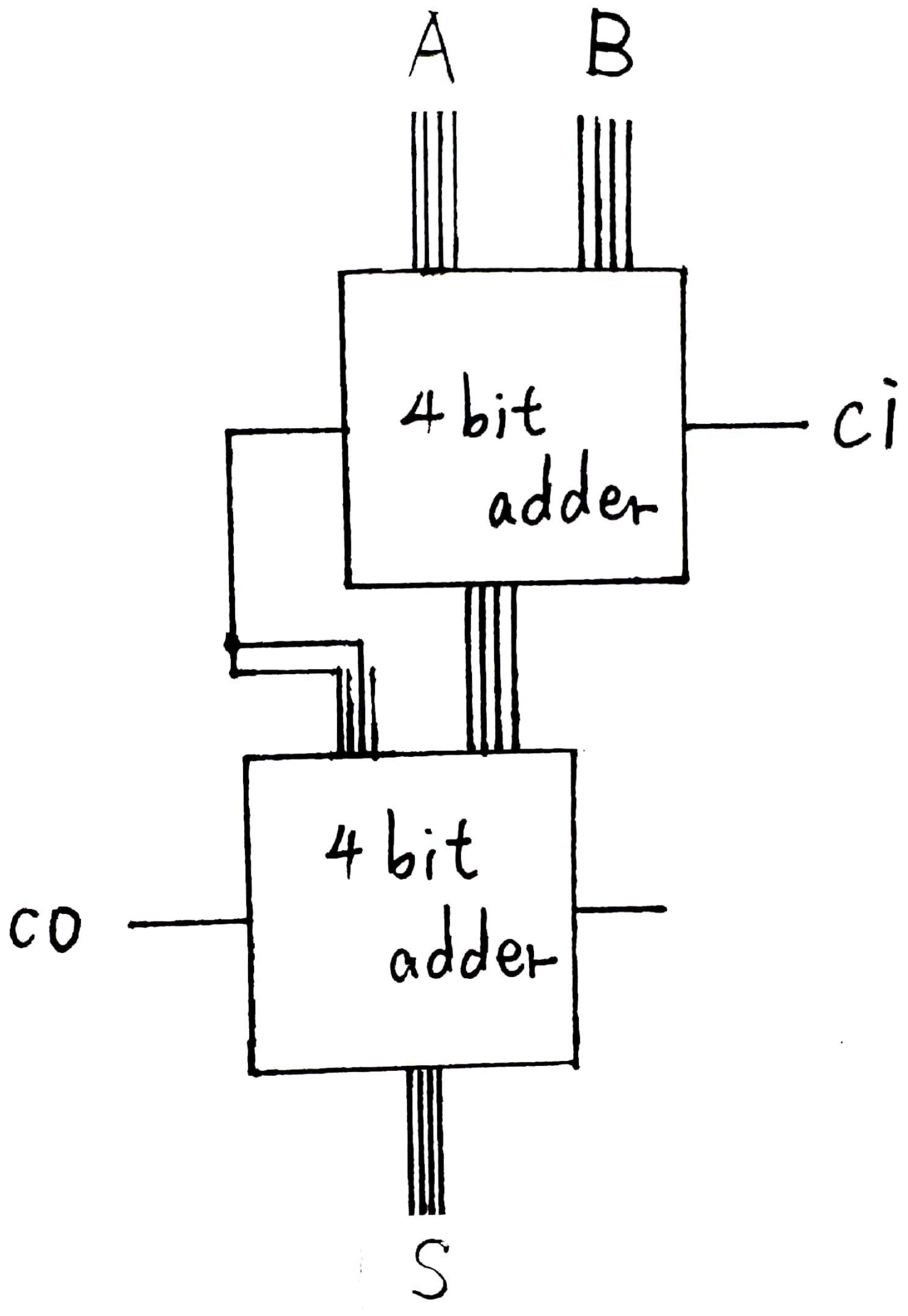
```
module larger(
    output  [2:0]   larger,
    input   [2:0]   a,        // (a2a1a0)
    input   [2:0]   b         // (b2b1b0)
    );
    wire w1;                  // connect (a>b) and sel

    // comparator
    assign w1 =               // (a>b)
        (a[2] & ~b[2]) |
        (a[1] & ~b[1]) & (a[2] ~^ b[2]) |
        (a[0] & ~b[0]) & (a[1] ~^ b[1]) & (a[2] ~^ b[2]);

    // MUX
    assign larger[2] = w1 & a[2] | ~w1 & b[2];
    assign larger[1] = w1 & a[1] | ~w1 & b[1];
    assign larger[0] = w1 & a[0] | ~w1 & b[0];

endmodule                     // larger
```

Write the test bench and verify the design.

```
module larger_test();
    // output
    wire    [2:0]   larger;

    // input
    reg     [2:0]   a;        // (a2a1a0)
    reg     [2:0]   b;        // (b2b1b0)

    // DUT
    larger  U0(
        .larger(larger),
        .a(a),                // (a2a1a0)
        .b(b)                 // (b2b1b0)
        );

    // stimulus
    always  #10 {a, b} = {a, b} + 1;
    initial     {a, b} = 0;

endmodule                     // larger_test
```



4 (Bonus) Design a single digit decimal adder with input A(a3a2a1a0), B(b3b2b1b0), Cin(ci), and output S(s3s2s1s0) and Cout(co).


A
B
4 bit adder
ci
4 bit adder
CO
S