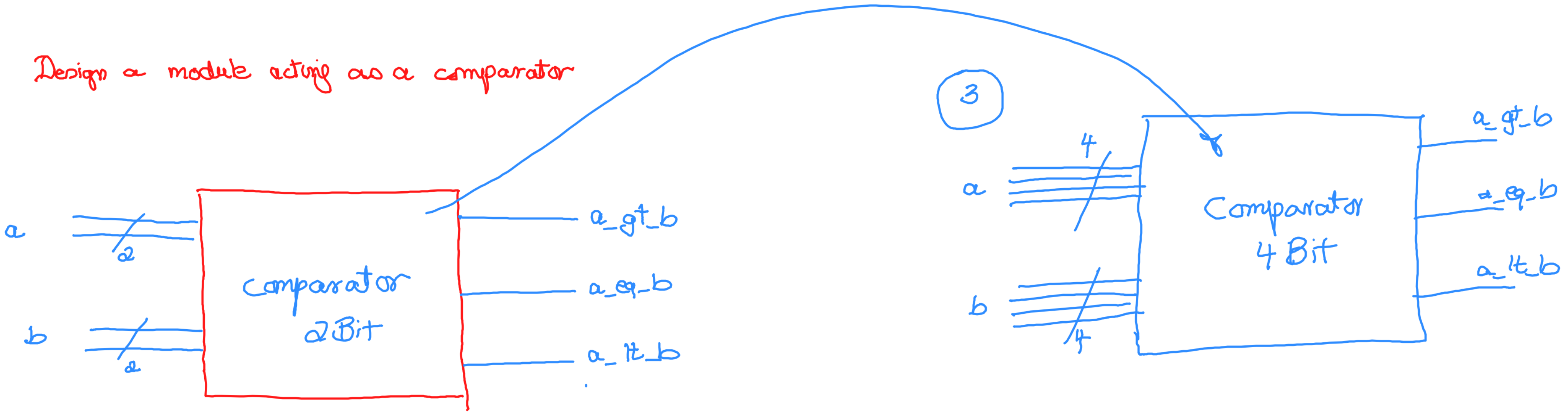
Design a module acting as a comparator



① ⊛ Gate level design (K-map , write the Switching function)
assign

② ⊛ Design using Behavioral description.

(a>b)

(a==b)

(a<b)

conditional operator

if the condition is false

```
assign x = (a>b) ? 1'b1 : 1'b0 ;
```

if the condition is true
x = 1

① Comparator Gate level Design :

	B_1B_0 \ A_1A_0	0 00	1 01	3 11	2 10
0	00	1	0	0	0
1	01	0	1	0	0
3	11	0	0	1	0
2	10	0	0	0	1

a_eq_b

	B_1B_0 \ A_1A_0	0 00	1 01	3 11	2 10
0	00	0	1	1	1
1	01	0	0	1	1
3	11	0	0	0	0
2	10	0	0	1	0

a_gt_b

	B_1B_0 \ A_1A_0	0 00	1 01	3 11	2 10
0	00	0	0	0	0
1	01	1	0	0	0
3	11	1	1	0	1
2	10	1	1	0	0

a_lt_b

$$a\_eq\_b = \underline{\bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0} + \underline{\bar{A}_1A_0\bar{B}_1B_0} + \underline{A_1A_0B_1B_0} + \underline{A_1\bar{A}_0B_1\bar{B}_0}$$

$$a\_gt\_b = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$$

$$a\_lt\_b = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$$

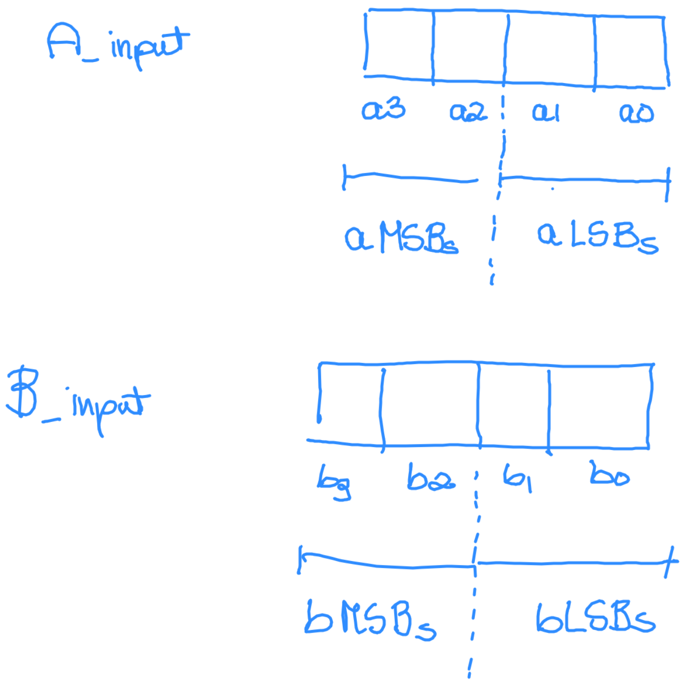
```
module   TwoBit_comparator ( output  a_eq_b , a_lt_b , a_gt_b ,
                             input  [1:0]  a, b  ) ;

wire   A0bar ,  A1 bar ,  B1 bar , B0 bar ;
wire   c1 , c2 , c3 , c4  ;

Not    not1    ( A0bar , a[0] ) ;
NOT    not2    ( A1bar , a[1] ) ;
NOT    not 3   ( B0bar , b[0] ) ;
NOT    not4    ( B1bar , b[1] ) ;
AND    and1    (c1 , A0bar, A1bar , B0bar, B1bar) ;
AND    and2    (c2 , A1 bar, B1bar , a[0] , b[0]) ;
AND    and3    (c3 , A0bar, B0bar , a[1] , b[1]) ;
AND    and4    (c4 , a[1] , a[0] , b[1] , b[0]) ;

OR     or1  (a_eq_b , c1 , c2 , c3 , c4) ;

complete the code for
a_lt_b
a_gt_b

endmodule
```

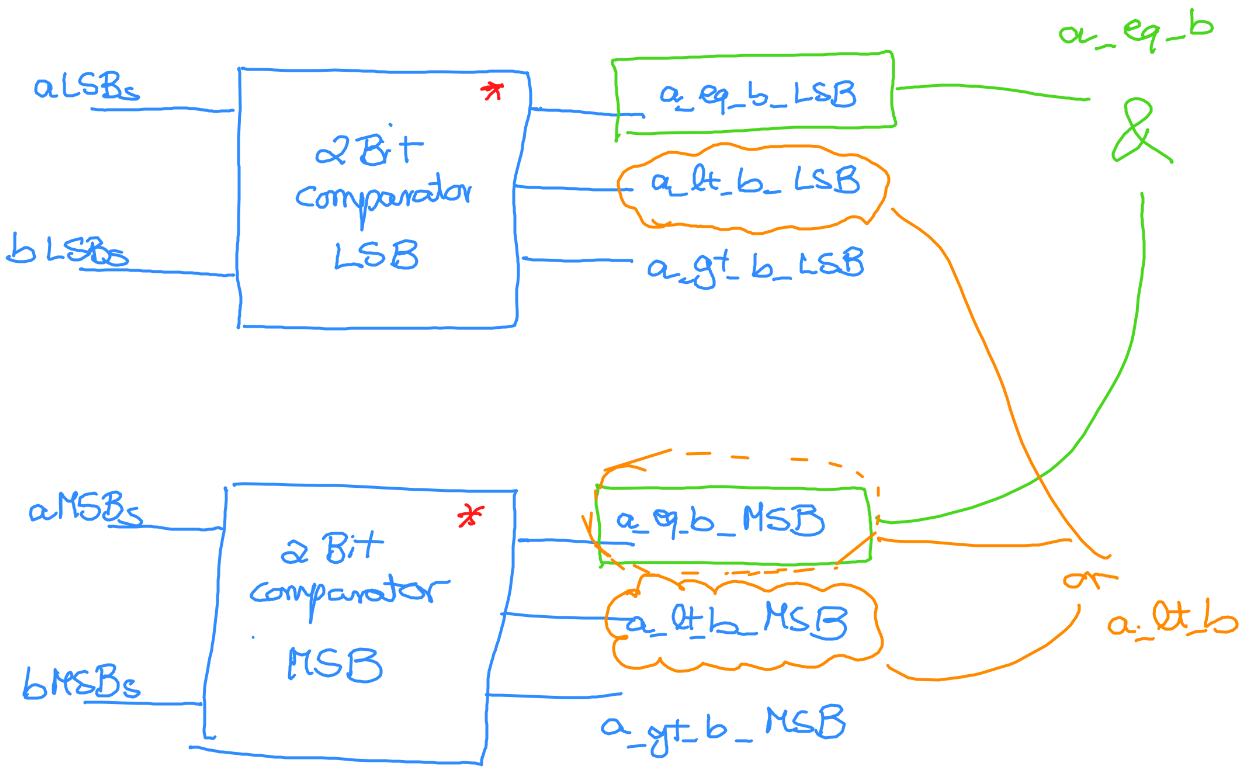
```
module   TwoBit_comparator  ( output   a_gt_b , a_eq_b , a_lt_b ,
                              input  [1:0]  a, b ) ;

wire  A0 , A1 , B1 , B2 ;

assign   A0  =  a [0]  ;
assign   A1  =  a [1]  ;
assign   B0  =  b [0]  ;
assign   B1  =  b [1]  ;
                         not         and
assign   a_eq_b  =  ( (~A1) & (~A0) & (~B1) & (~B0) )      //  00 & 00
                  | ( (~A1) &  A0   & (~B1) &  B0  )       //  01 & 01
          OR      | (  A1 & (~A0)   &  B1   & (~B0) )      //  10 & 10
                  | (  A1 & A0      &  B1   &  B0  ) ;     //  11 & 11
```

```
assign   a_gt_b  =  (A1 & (~B1)) | (A1 & A0 & (~B0)) | (A0 & (~B0) & (~B1) ;
assign   a_lt_b  =  ((~A1) & B1 ) | ((~A) & (~A0) & B0) | (B1 & B0 & (~A0) ;

endmodule
```

```
module  TwoBit_comparator ( output  a_eq_b , a_lt_b , a_gt_b ;
                            input  [1:0]  a , b ) ;

assign a_eq_b  =  (a == b) ?  1'b1  :  1'b0 ;
assign a_lt_b  =  (a < b ) ?  1'b1  :  1'b0 ;
assign a_gt_b  =  (a > b)  ?  1'b1  :  1'b0 ;

   endmodule
```

4Bit_ comparator





```
● a_eq_b = a_eq_b_LSB & a_eq_b_MSB ;
  a_lt_b = a_lt_b_MSB | (a_eq_b_MSB & a_lt_b_LSB);
  a_gt_b = a_gt_b_MSB | (a_eq_b_MSB & a_gt b_LSB) ;
```