```
/*      EXOR    */
module  EXOR    ( IN1, IN2, OUT );
input   IN1, IN2;
output  OUT;
wire    NOT1,NOT2,AND1,AND2;
        not     U1      ( NOT1, IN1 ),
                U2      ( NOT2, IN2 );
        and     U3      ( AND1, NOT1, IN2 ),
                U4      ( AND2, NOT2, IN1 );
        or      U5      ( OUT , AND1, AND2 );
endmodule
```

```
/*      EXOR    */
module  EXOR    ( IN1, IN2, OUT );
input   IN1, IN2;
output  OUT;
        xor     U1      ( OUT, IN1, IN2 );
endmodule
```

```
/*      EXOR    */
module  EXOR    ( IN1, IN2, OUT );
input   IN1, IN2;
output  OUT;
        assign  OUT = ~IN1 & IN2 | IN1 & ~IN2;
endmodule
```

```
/*      EXOR    */
module  EXOR    ( IN1, IN2, OUT );
input   IN1, IN2;
output  OUT;
        assign  OUT = IN1 ^ IN2;
endmodule
```

```
/*      EXOR    */
module  EXOR    ( IN1, IN2, OUT );
input   IN1, IN2;
output  OUT;
        assign  OUT = EXOR_FUNC ( IN1, IN2 );

function        EXOR_FUNC;
input   IN1, IN2;
        if      ( IN1 ^ IN2 )
                EXOR_FUNC = 1;
        else
                EXOR_FUNC = 0;
endfunction

endmodule
```

```
/*      2-1 SELECTOR    */
module  SEL     ( A, B, SEL, OUT );
input   A, B, SEL;
output  OUT;
wire    SEL_NOT, AND1, AND2;
        not     U1      ( SEL_NOT, SEL );
        and     U2      ( AND1, B, SEL ),
                U3      ( AND2, A, SEL_NOT );
        or      U4      ( OUT , AND1, AND2 );
endmodule
```

```
/*      4-1 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   A, B, C, D;
input   [1:0] SEL;
output  OUT;
wire    SEL1_NOT, SEL0_NOT, AND1, AND2, AND3, AND4;
        not     U1      ( SEL1_NOT, SEL[1] ),
                U2      ( SEL0_NOT, SEL[0] );
        and     U3      ( AND1, A, SEL1_NOT, SEL0_NOT ),
                U4      ( AND2, B, SEL1_NOT, SEL[0] ),
                U5      ( AND3, C, SEL[1]  , SEL0_NOT ),
                U6      ( AND4, D, SEL[1]  , SEL[0] );
        or      U7      ( OUT , AND1, AND2, AND3, AND4 );
endmodule
```

```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   [1:0] A, B, C, D, SEL;
output  [1:0]OUT;
wire    SEL1_NOT, SEL0_NOT;
wire    AND_A1, AND_B1, AND_C1, AND_D1;
wire    AND_A0, AND_B0, AND_C0, AND_D0;
        not     U1      ( SEL1_NOT, SEL[1] ),
                U2      ( SEL0_NOT, SEL[0] );
        and     UAND_A1 ( AND_A1, A[1], SEL1_NOT, SEL0_NOT ),
                UAND_B1 ( AND_B1, B[1], SEL1_NOT, SEL[0] ),
                UAND_C1 ( AND_C1, C[1], SEL[1]  , SEL0_NOT ),
                UAND_D1 ( AND_D1, D[1], SEL[1]  , SEL[0] );
        and     UAND_A0 ( AND_A0, A[0], SEL1_NOT, SEL0_NOT ),
                UAND_B0 ( AND_B0, B[0], SEL1_NOT, SEL[0] ),
                UAND_C0 ( AND_C0, C[0], SEL[1]  , SEL0_NOT ),
                UAND_D0 ( AND_D0, D[0], SEL[1]  , SEL[0] );
        or      OUT1    ( OUT[1] , AND_A1, AND_B1, AND_C1, AND_D1 ),
                OUT0    ( OUT[0] , AND_A0, AND_B0, AND_C0, AND_D0 );
endmodule
```

```
/*      2-1 SELECTOR    */
module  SEL     ( A, B, SEL, OUT );
input   A, B, SEL;
output  OUT;

        assign  OUT = ~SEL & A | SEL & B;

endmodule
```

```
/*      4-1 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   A, B, C, D;
input   [1:0] SEL;
output  OUT;

        assign  OUT = ~SEL[1] & ~SEL[0] & A
                    | ~SEL[1] &  SEL[0] & B
                    |  SEL[1] & ~SEL[0] & C
                    |  SEL[1] &  SEL[0] & D;

endmodule
```

```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   [1:0] A, B, C, D, SEL;
output  [1:0]OUT;

        assign  OUT[1] = ~SEL[1] & ~SEL[0] & A[1]
                       | ~SEL[1] &  SEL[0] & B[1]
                       |  SEL[1] & ~SEL[0] & C[1]
                       |  SEL[1] &  SEL[0] & D[1];

        assign  OUT[0] = ~SEL[1] & ~SEL[0] & A[0]
                       | ~SEL[1] &  SEL[0] & B[0]
                       |  SEL[1] & ~SEL[0] & C[0]
                       |  SEL[1] &  SEL[0] & D[0];

endmodule
```

```
/*      2-1 SELECTOR    */
module  SEL     ( A, B, SEL, OUT );
input   A, B, SEL;
output  OUT;

        assign  OUT = ( SEL == 0 )? A: B;

endmodule
```

```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   A, B, C, D;
input   [1:0] SEL;
output  OUT;

        assign  OUT = ( SEL[1] == 0 )?
                 (( SEL[0] == 0 )? A: B ):(( SEL[0] == 0 )? C: D );

endmodule
```

```
/*      2-1 SELECTOR    */
module  SEL     ( A, B, SEL, OUT );
input   A, B, SEL;
output  OUT;

        assign  OUT = SEL2_1_FUNC ( A, B, SEL );

function        SEL2_1_FUNC;
input   A, B, SEL;
        if ( SEL == 0 )
                        SEL2_1_FUNC = A;
                else
                        SEL2_1_FUNC = B;
endfunction

endmodule
```

```
/*      2-1 SELECTOR    */
module  SEL     ( A, B, SEL, OUT );
input   A, B, SEL;
output  OUT;

        assign  OUT = SEL2_1_FUNC ( A, B, SEL );

function        SEL2_1_FUNC;
input   A, B, SEL;
        case ( SEL )
                0:SEL2_1_FUNC = A;
                1:SEL2_1_FUNC = B;
        endcase
endfunction

endmodule
```

```
/*      4-1 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   A, B, C, D;
input   [1:0] SEL;
output  OUT;

        assign  OUT = SEL4_1_FUNC ( A, B, C, D, SEL );

function        SEL4_1_FUNC;
input   A, B, C, D;
input   [1:0] SEL;
        if ( SEL[1] == 0 )
                if ( SEL[0] == 0 )
                        SEL4_1_FUNC = A;
                else
                        SEL4_1_FUNC = B;
        else
                if ( SEL[0] == 0 )
                        SEL4_1_FUNC = C;
                else
                        SEL4_1_FUNC = D;
endfunction

endmodule
```

```
/*      4-1 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   A, B, C, D;
input   [1:0] SEL;
output  OUT;

        assign  OUT = SEL4_1_FUNC ( A, B, C, D, SEL );

function        SEL4_1_FUNC;
input   A, B, C, D;
input   [1:0] SEL;
        case ( SEL )
                0:SEL4_1_FUNC = A;
                1:SEL4_1_FUNC = B;
                2:SEL4_1_FUNC = C;
                3:SEL4_1_FUNC = D;
        endcase
endfunction

endmodule
```

```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   [1:0] A, B, C, D, SEL;
output  [1:0] OUT;

        assign  OUT = SEL4_2_FUNC ( A, B, C, D, SEL );

function        [1:0] SEL4_2_FUNC;
input   [1:0] A, B, C, D;
input   [1:0] SEL;
        if ( SEL[1] == 0 )
                if ( SEL[0] == 0 )
                        SEL4_2_FUNC = A;
                else
                        SEL4_2_FUNC = B;
        else if ( SEL[0] == 0 )
                        SEL4_2_FUNC = C;
                else
                        SEL4_2_FUNC = D;
endfunction

endmodule
```

```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   [1:0] A, B, C, D, SEL;
output  [1:0] OUT;

        assign  OUT = SEL4_2_FUNC ( A, B, C, D, SEL );

function        [1:0] SEL4_2_FUNC;
input   [1:0] A, B, C, D;
input   [1:0] SEL;
        case ( SEL )
                0:SEL4_2_FUNC = A;
                1:SEL4_2_FUNC = B;
                2:SEL4_2_FUNC = C;
                3:SEL4_2_FUNC = D;
        endcase
endfunction

endmodule
```

```
/*      2bit COMPARATOR */
module  COMP    ( X, Y, LG, EQ, SM );
input   [1:0] X, Y;
output  LG, EQ, SM;

        assign  LG = X[0] & ~Y[1] & ~Y[0]
                        | X[1] & X[0] & ~Y[0]
                        | X[1] & ~Y[1];
        assign  EQ = ( X[1] ~^ Y[1] ) & ( X[0] ~^ Y[0] );
        assign  SM = ~X[0] & Y[1] & Y[0]
                        | ~X[1] & ~X[0] & Y[0]
                        | ~X[1] & Y[1];

endmodule
```

```
/*      2bit COMPARATOR */
module  COMP    ( X, Y, LG, EQ, SM );
input   [1:0] X, Y;
output  LG, EQ, SM;
        assign  { LG, EQ, SM } = FUNC_COMP ( X, Y );

function         [2:0] FUNC_COMP;
input   [1:0] X, Y;
        if ( X > Y )
                        FUNC_COMP = 3'b100;
        else    if ( X < Y )
                        FUNC_COMP = 3'b001;
                else
                        FUNC_COMP = 3'b010;
endfunction

endmodule
```

```
/*      4bit COMPARATOR */
module  COMP    ( X, Y, LG_OUT, EQ_OUT, SM_OUT );
input   [3:0] X, Y;
output  LG_OUT, EQ_OUT, SM_OUT;
wire    [2:0] LG, EQ, SM;
        FULL_COMP       COMP0   ( X[0], Y[0], 1'b0, 1'b1, 1'b0,
                                  LG[0], EQ[0], SM[0] ),
                        COMP1   ( X[1], Y[1], LG[0], EQ[0], SM[0],
                                  LG[1], EQ[1], SM[1] ),
                        COMP2   ( X[2], Y[2], LG[1], EQ[1], SM[1],
                                  LG[2], EQ[2], SM[2] ),
                        COMP3   ( X[3], Y[3], LG[2], EQ[2], SM[2],
                                  LG_OUT, EQ_OUT, SM_OUT );
endmodule


/*      FULL COMPARATOR */
module  FULL_COMP       ( X, Y, LG_IN, EQ_IN, SM_IN, LG_OUT, EQ_OUT, SM_OUT );
input   X, Y;
input   LG_IN, EQ_IN, SM_IN;
output  LG_OUT, EQ_OUT, SM_OUT;
        assign  { LG_OUT, EQ_OUT, SM_OUT }
                        = FUNC_COMP ( X, Y, LG_IN, EQ_IN, SM_IN );

function        [2:0] FUNC_COMP;
input   X, Y;
input   LG_IN, EQ_IN, SM_IN;
        if ( X > Y )
                                        FUNC_COMP = 3'b100;
        else    if ( X < Y )
                                        FUNC_COMP = 3'b001;
                else    if ( LG_IN )
                                        FUNC_COMP = 3'b100;
                        else    if ( SM_IN )
                                        FUNC_COMP = 3'b001;
                                else    FUNC_COMP = 3'b010;
endfunction

endmodule
```

```
/*      4bit COMPARATOR */
module  COMP    ( X, Y, LG, EQ, SM );
input   [3:0] X, Y;
output  LG, EQ, SM;
        assign  { LG, EQ, SM } = FUNC_COMP ( X, Y );

function         [2:0] FUNC_COMP;
input   [3:0] X, Y;
        if ( X > Y )
                        FUNC_COMP = 3'b100;
        else    if ( X < Y )
                        FUNC_COMP = 3'b001;
                else
                        FUNC_COMP = 3'b010;
endfunction

endmodule
```

```
/*  Data Difinision     */
`define         SW_IN0  10'b0000000001
`define         SW_IN1  10'b0000000010
`define         SW_IN2  10'b0000000100
`define         SW_IN3  10'b0000001000
`define         SW_IN4  10'b0000010000
`define         SW_IN5  10'b0000100000
`define         SW_IN6  10'b0001000000
`define         SW_IN7  10'b0010000000
`define         SW_IN8  10'b0100000000
`define         SW_IN9  10'b1000000000

/*      ENCORDER        */
module  ENC     ( IN, OUT );
input   [9:0]   IN;
output  [3:0]   OUT;
        assign OUT = FUNC_ENC ( IN );

function        [3:0] FUNC_ENC;
input   [9:0]   IN;
        case ( IN )
                `SW_IN0:FUNC_ENC = 0;
                `SW_IN1:FUNC_ENC = 1;
                `SW_IN2:FUNC_ENC = 2;
                `SW_IN3:FUNC_ENC = 3;
                `SW_IN4:FUNC_ENC = 4;
                `SW_IN5:FUNC_ENC = 5;
                `SW_IN6:FUNC_ENC = 6;
                `SW_IN7:FUNC_ENC = 7;
                `SW_IN8:FUNC_ENC = 8;
                `SW_IN9:FUNC_ENC = 9;
                default:FUNC_ENC = 15;
        endcase
endfunction

endmodule
```

```
`define         OUT_0   10'b00_0000_0001
`define         OUT_1   10'b00_0000_0010
`define         OUT_2   10'b00_0000_0100
`define         OUT_3   10'b00_0000_1000
`define         OUT_4   10'b00_0001_0000
`define         OUT_5   10'b00_0010_0000
`define         OUT_6   10'b00_0100_0000
`define         OUT_7   10'b00_1000_0000
`define         OUT_8   10'b01_0000_0000
`define         OUT_9   10'b10_0000_0000
`define         OUT_ERR 10'b00_0000_0000

/*      DECORDER        */
module  DEC     ( IN, OUT, ERR );
input   [3:0]   IN;
output  [9:0]   OUT;
output  ERR;
        assign  { ERR, OUT } = FUNC_DEC ( IN );

function        [10:0] FUNC_DEC;
input   [3:0]   IN;
        case ( IN )
                0:FUNC_DEC = { 1'b0, `OUT_0 };
                1:FUNC_DEC = { 1'b0, `OUT_1 };
                2:FUNC_DEC = { 1'b0, `OUT_2 };
                3:FUNC_DEC = { 1'b0, `OUT_3 };
                4:FUNC_DEC = { 1'b0, `OUT_4 };
                5:FUNC_DEC = { 1'b0, `OUT_5 };
                6:FUNC_DEC = { 1'b0, `OUT_6 };
                7:FUNC_DEC = { 1'b0, `OUT_7 };
                8:FUNC_DEC = { 1'b0, `OUT_8 };
                9:FUNC_DEC = { 1'b0, `OUT_9 };
                default :FUNC_DEC = { 1'b1, `OUT_ERR };
        endcase
endfunction

endmodule
```

```
/*  4-10 DECORDER        */
module DEC      ( IN, OUT, ERR );
input   [3:0]   IN;
output  [9:0]   OUT;
output  ERR;

        assign  { ERR, OUT } = FUNC_OUT ( IN );

function        [10:0] FUNC_OUT;
input   [3:0]   IN;
        begin
            FUNC_OUT = 0;
            case ( IN )
                0       : FUNC_OUT[0]  = 1;
                1       : FUNC_OUT[1]  = 1;
                2       : FUNC_OUT[2]  = 1;
                3       : FUNC_OUT[3]  = 1;
                4       : FUNC_OUT[4]  = 1;
                5       : FUNC_OUT[5]  = 1;
                6       : FUNC_OUT[6]  = 1;
                7       : FUNC_OUT[7]  = 1;
                8       : FUNC_OUT[8]  = 1;
                9       : FUNC_OUT[9]  = 1;
                default : FUNC_OUT[10] = 1;
            endcase
        end
endfunction
endmodule
```

```
`define         SEG_OUT_0       7'b011_1111
`define         SEG_OUT_1       7'b000_0110
`define         SEG_OUT_2       7'b101_1011
`define         SEG_OUT_3       7'b100_1111
`define         SEG_OUT_4       7'b110_0110
`define         SEG_OUT_5       7'b110_1101
`define         SEG_OUT_6       7'b111_1101
`define         SEG_OUT_7       7'b010_0111
`define         SEG_OUT_8       7'b111_1111
`define         SEG_OUT_9       7'b110_1111
`define         SEG_OUT_ERR     7'b111_1001

/*      7SEG_DECORDER   */
module  SEG7_DEC        ( IN, OUT );
input   [3:0]   IN;
output  [6:0]   OUT;
        assign  OUT = FUNC_SEG7_DEC ( IN );

function        [6:0] FUNC_SEG7_DEC;
input   [3:0]   IN;
        case ( IN )
                0:      FUNC_SEG7_DEC = `SEG_OUT_0;
                1:      FUNC_SEG7_DEC = `SEG_OUT_1;
                2:      FUNC_SEG7_DEC = `SEG_OUT_2;
                3:      FUNC_SEG7_DEC = `SEG_OUT_3;
                4:      FUNC_SEG7_DEC = `SEG_OUT_4;
                5:      FUNC_SEG7_DEC = `SEG_OUT_5;
                6:      FUNC_SEG7_DEC = `SEG_OUT_6;
                7:      FUNC_SEG7_DEC = `SEG_OUT_7;
                8:      FUNC_SEG7_DEC = `SEG_OUT_8;
                9:      FUNC_SEG7_DEC = `SEG_OUT_9;
                default:FUNC_SEG7_DEC = `SEG_OUT_ERR;
        endcase
endfunction

endmodule
```

```
/*      EVEN_PARITY_GENERATOR   */
module  EVEN_PARITY_GEN ( IN, ODD_OUT );
input   [3:0]   IN;
output  ODD_OUT;
        assign  ODD_OUT = IN[3] ^ IN[2] ^ IN[1] ^ IN[0];
endmodule
```

```
/*      EVEN_PARITY_GENERATOR   */
module  EVEN_PARITY_GEN ( IN, ODD_OUT );
input   [3:0]   IN;
output  ODD_OUT;
        assign  ODD_OUT = ^ ( IN );
endmodule
```

```
/*      EVEN_PARITY_GENERATOR   */
module  EVEN_PARITY_GEN ( IN, ODD_OUT );
input   [3:0]   IN;
output  ODD_OUT;
        assign  ODD_OUT = FUNC_GEN ( IN );

function        FUNC_GEN;
input   [3:0]   IN;
integer i;
        begin
                FUNC_GEN = 0;
                for     ( i = 0; i <= 3; i = i + 1 )
                        FUNC_GEN = FUNC_GEN ^ IN[ i ];
        end
endfunction

endmodule
```

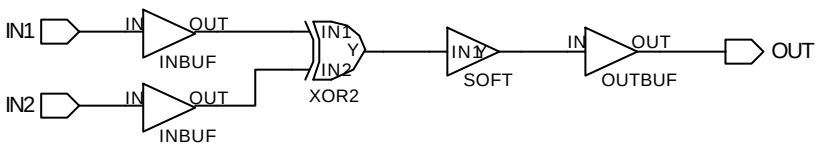
```
/*      4-2 SELECTOR    */
module  SEL     ( A, B, C, D, SEL, OUT );
input   [1:0] A, B, C, D;
input   [1:0] SEL;
output  [1:0] OUT;

        assign  OUT = ( SEL[1] == 0 )?
                 (( SEL[0] == 0 )? A: B ):(( SEL[0] == 0 )? C: D );

endmodule
```

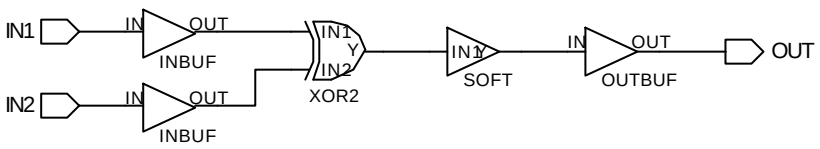
```
/*      4bit COMPARATOR */
module  COMP    ( X, Y, LG_OUT, EQ_OUT, SM_OUT );
input   [3:0] X, Y;
output  LG_OUT, EQ_OUT, SM_OUT;
wire    [2:0] LG, EQ, SM;
        FULL_COMP       COMP0   ( X[0], Y[0], 1'b0, 1'b1, 1'b0,
                                  LG[0], EQ[0], SM[0] ),
                        COMP1   ( X[1], Y[1], LG[0], EQ[0], SM[0],
                                  LG[1], EQ[1], SM[1] ),
                        COMP2   ( X[2], Y[2], LG[1], EQ[1], SM[1],
                                  LG[2], EQ[2], SM[2] ),
                        COMP3   ( X[3], Y[3], LG[2], EQ[2], SM[2],
                                  LG_OUT, EQ_OUT, SM_OUT );
endmodule


/*      FULL COMPARATOR */
module  FULL_COMP       ( X, Y, LG_IN, EQ_IN, SM_IN, LG_OUT, EQ_OUT, SM_OUT );
input   X, Y;
input   LG_IN, EQ_IN, SM_IN;
output  LG_OUT, EQ_OUT, SM_OUT;
        assign  LG_OUT = ~Y &  X | ( Y ~^ X ) & LG_IN;
        assign  SM_OUT =  Y & ~X | ( Y ~^ X ) & SM_IN;
        assign  EQ_OUT = ( Y ~^ X ) & EQ_IN;
endmodule
```

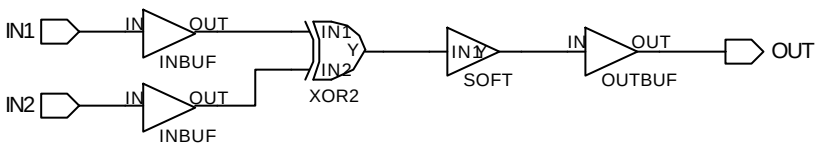
```
/*      ENCORDER        */
module  ENC     ( IN, OUT );
input   [9:0]   IN;
output  [3:0]   OUT;
        assign OUT[0] = IN[1] | IN[3] | IN[5] | IN[7] | IN[9];
        assign OUT[1] = IN[2] | IN[3] | IN[6] | IN[7];
        assign OUT[2] = IN[4] | IN[5] | IN[6] | IN[7];
        assign OUT[3] = IN[8] | IN[9];
endmodule
```

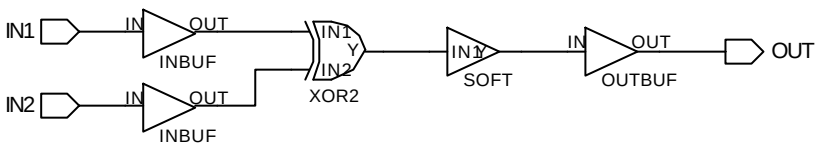
```
/*      ENCORDER        */
module  ENC     ( IN, OUT );
input   [9:0]   IN;
output  [3:0]   OUT;
        assign OUT = FUNC_ENC ( IN );

function        [3:0] FUNC_ENC;
input   [9:0]   IN;
integer i, j;
        begin
                j = 1;
                FUNC_ENC = 0;
                for ( i = 0; i <= 9; i = i + 1 )
                        begin
                                if ( IN & j )
                                        FUNC_ENC = FUNC_ENC | i;
                                j = j << 1;
                        end
        end
endfunction

endmodule
```

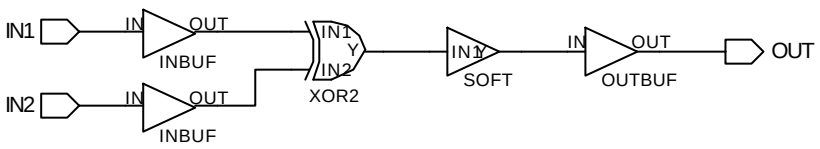
```
`define         OUT_0   8'b0000_0001
`define         OUT_1   8'b0000_0010
`define         OUT_2   8'b0000_0100
`define         OUT_3   8'b0000_1000
`define         OUT_4   8'b0001_0000
`define         OUT_5   8'b0010_0000
`define         OUT_6   8'b0100_0000
`define         OUT_7   8'b1000_0000

/*      DECORDER        */
module  DEC     ( IN, OUT );
input   [2:0]   IN;
output  [7:0]   OUT;
        assign  OUT = FUNC_DEC ( IN );

function        [7:0] FUNC_DEC;
input   [2:0]   IN;
        case ( IN )
                0       :FUNC_DEC = `OUT_0;
                1       :FUNC_DEC = `OUT_1;
                2       :FUNC_DEC = `OUT_2;
                3       :FUNC_DEC = `OUT_3;
                4       :FUNC_DEC = `OUT_4;
                5       :FUNC_DEC = `OUT_5;
                6       :FUNC_DEC = `OUT_6;
                default :FUNC_DEC = `OUT_7;
        endcase
endfunction

endmodule
```

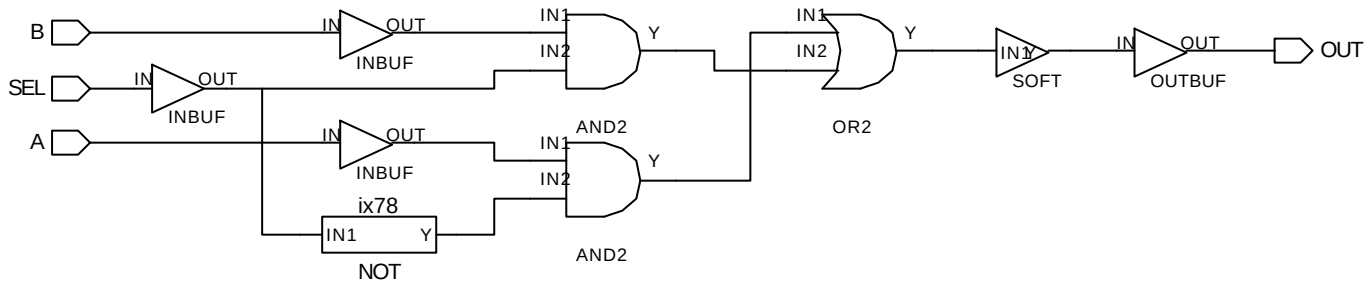

IN1
IN
OUT
INBUF
IN2
IN
OUT
INBUF
IN1
Y
IN2
XOR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

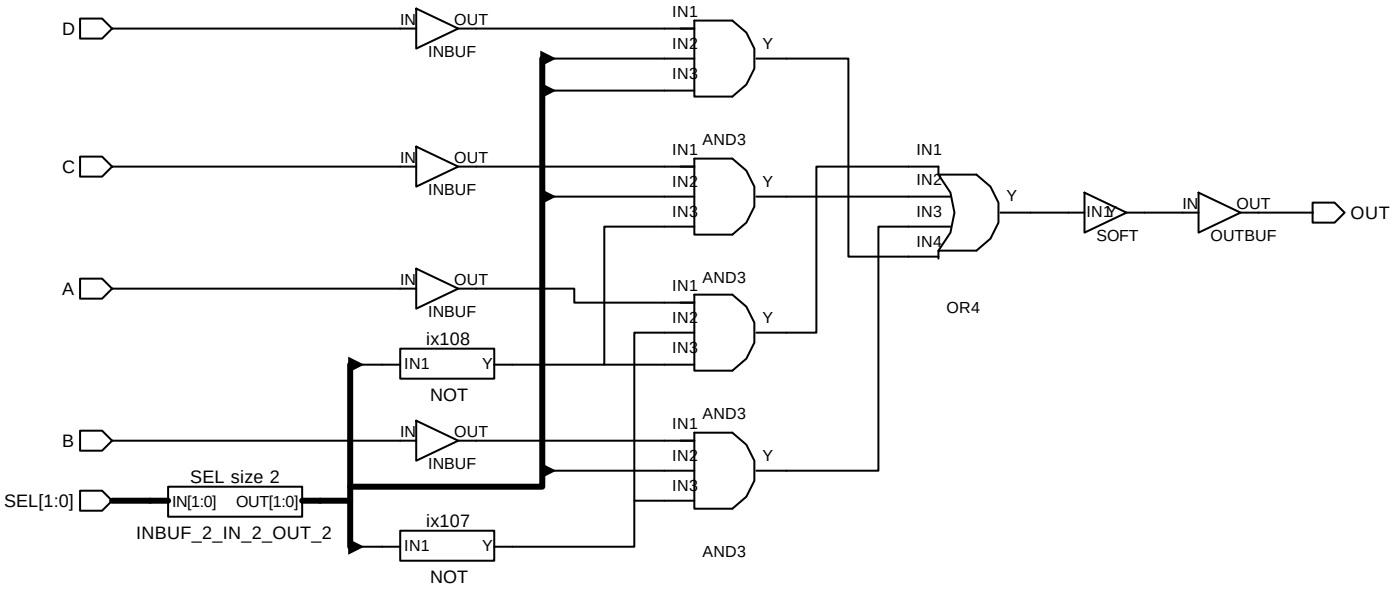

IN1
IN
OUT
INBUF
IN2
IN
OUT
INBUF
IN1
Y
IN2
XOR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

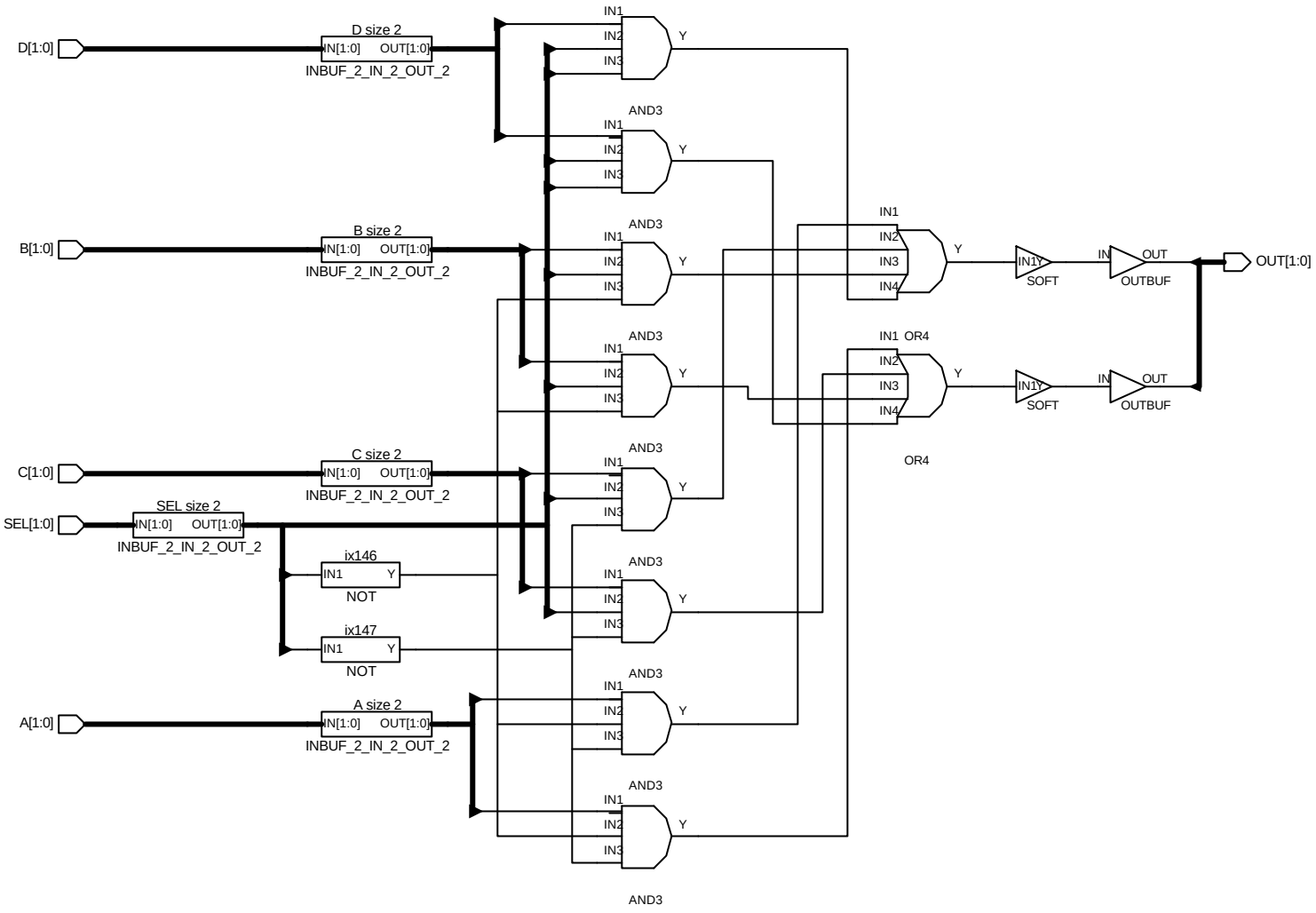

IN1
IN
OUT
INBUF
IN2
IN
OUT
INBUF
IN1
Y
IN2
XOR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

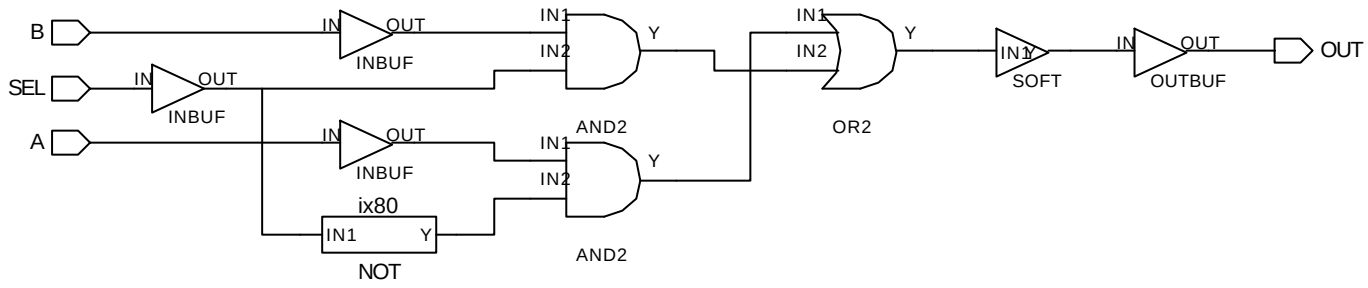

IN1
IN
OUT
INBUF
IN2
IN
OUT
INBUF
IN1
Y
IN2
XOR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

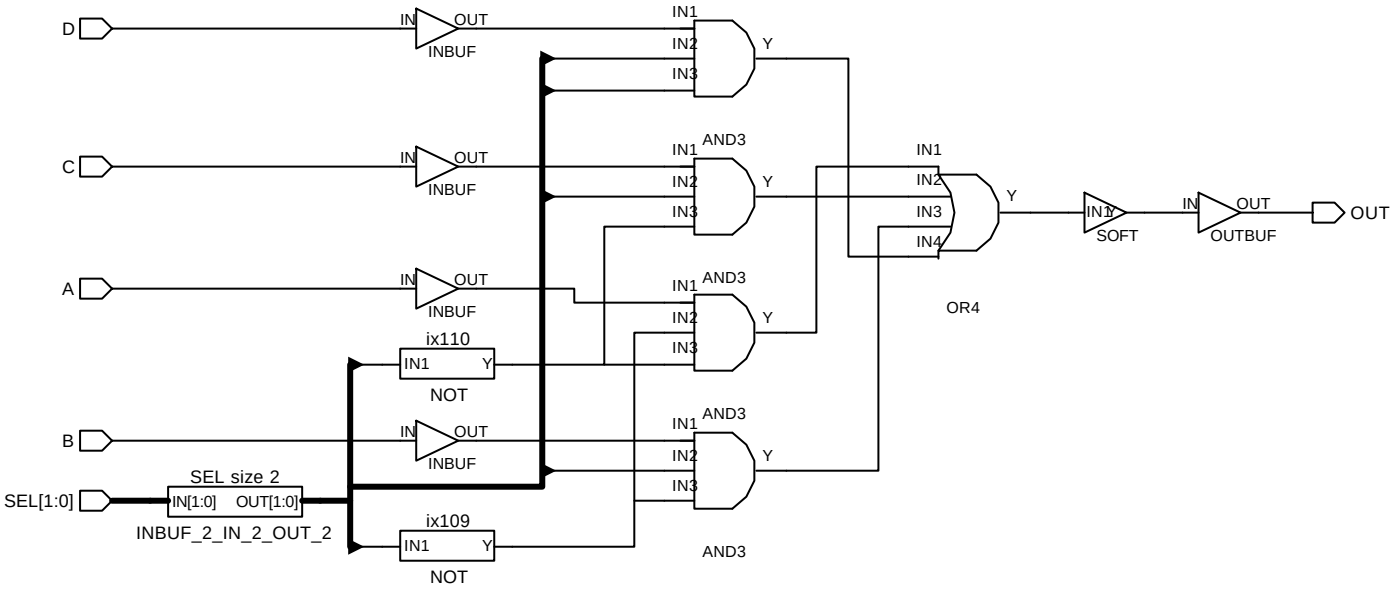

IN1
IN
OUT
INBUF
IN2
IN
OUT
INBUF
IN1
Y
IN2
XOR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

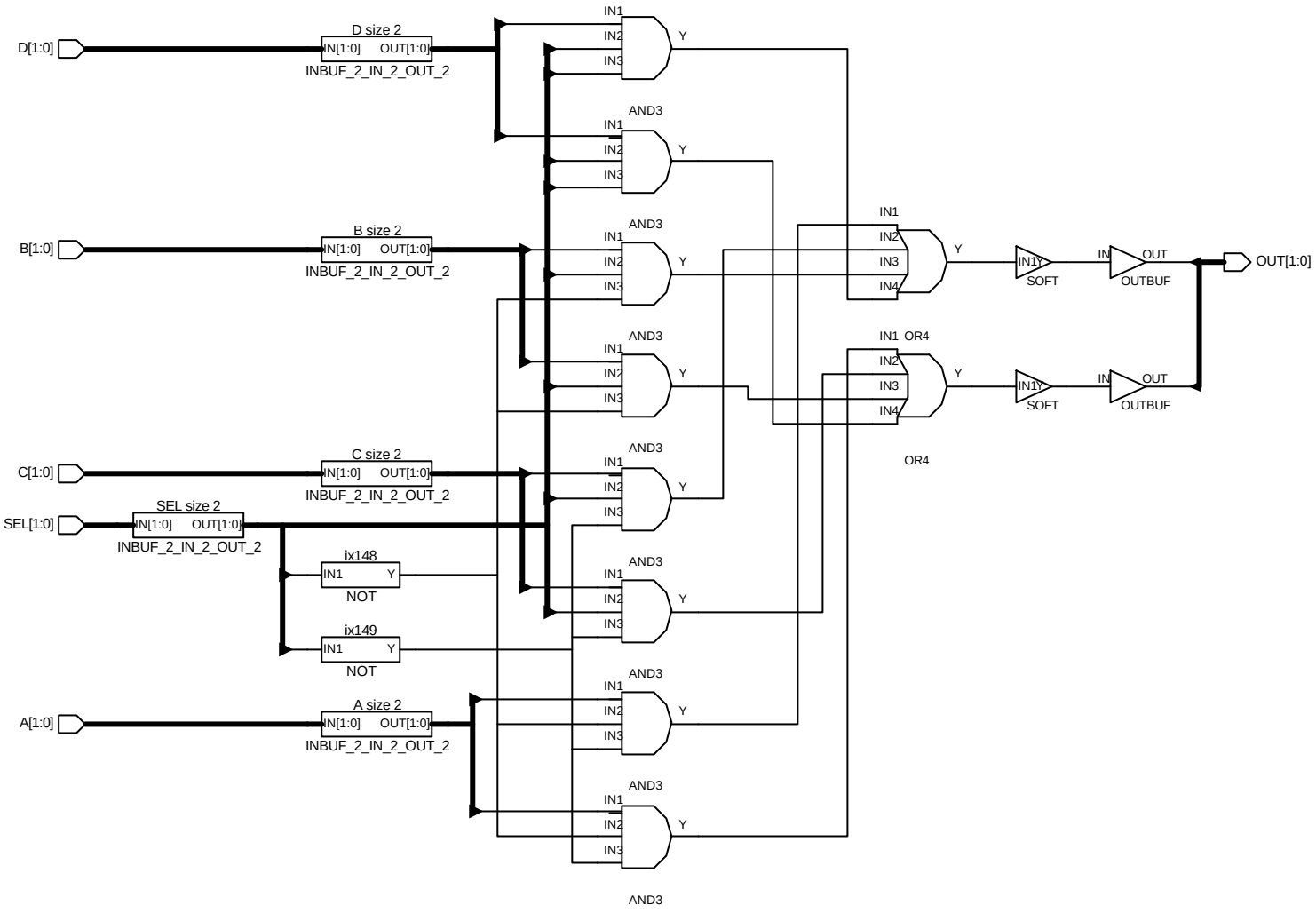

B
SEL
A
IN
OUT
INBUF
IN1
IN2
Y
AND2
OR2
SOFT
OUTBUF
OUT
ix78
NOT

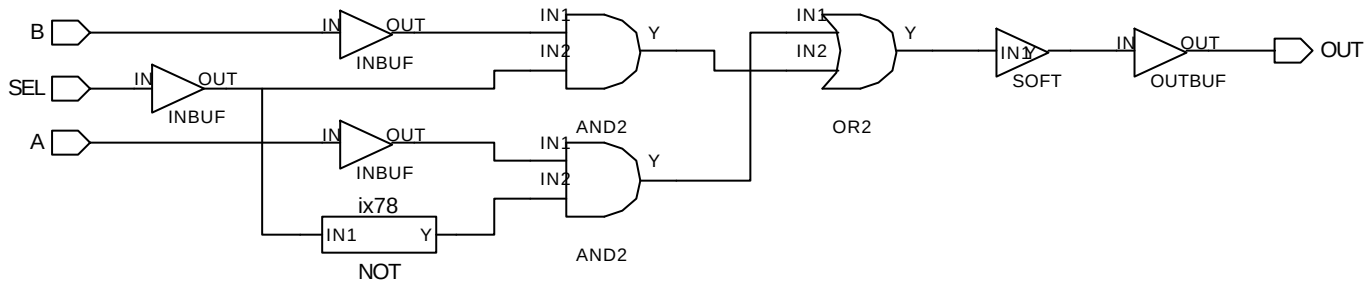

D
C
A
B
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
IN
OUT
INBUF
ix108
ix107
IN1
Y
NOT
AND3
IN2
IN3
IN4
OR4
SOFT
OUTBUF
OUT

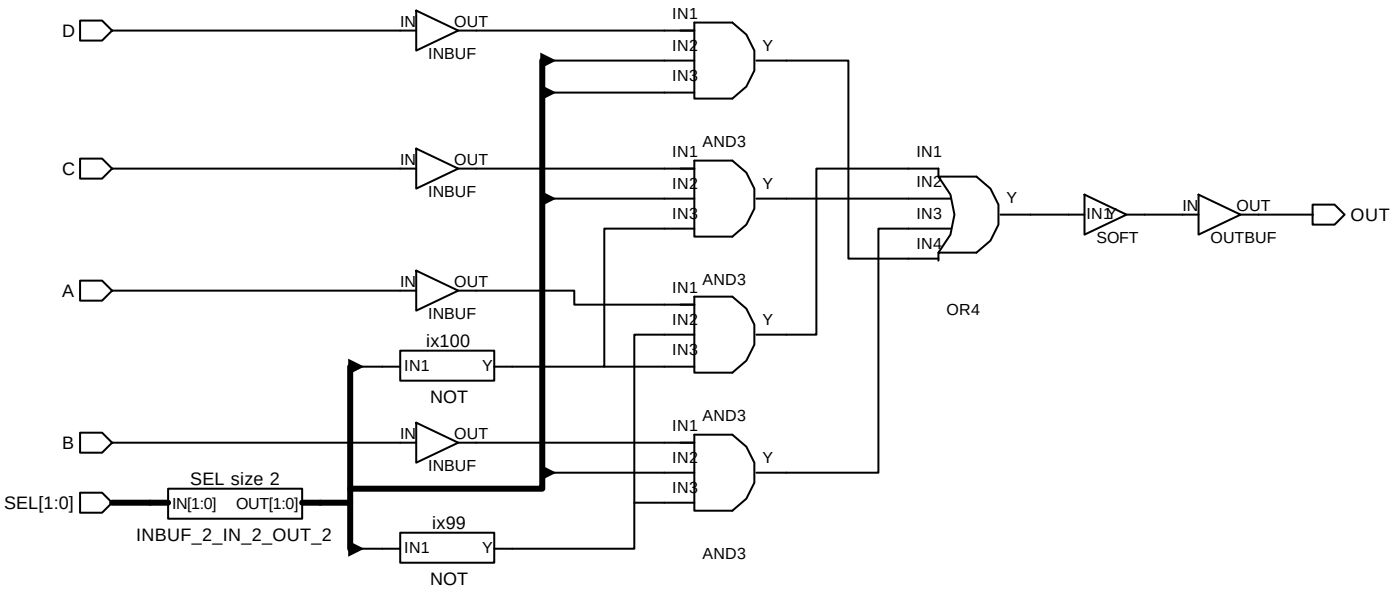

D[1:0]
D size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
B[1:0]
B size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
C[1:0]
C size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
A[1:0]
A size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix146
IN1 Y
NOT
ix147
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
IN1 OR4
IN2
IN3
IN4
Y
OR4
IN1Y
SOFT
IN
OUT
OUTBUF
OUT[1:0]

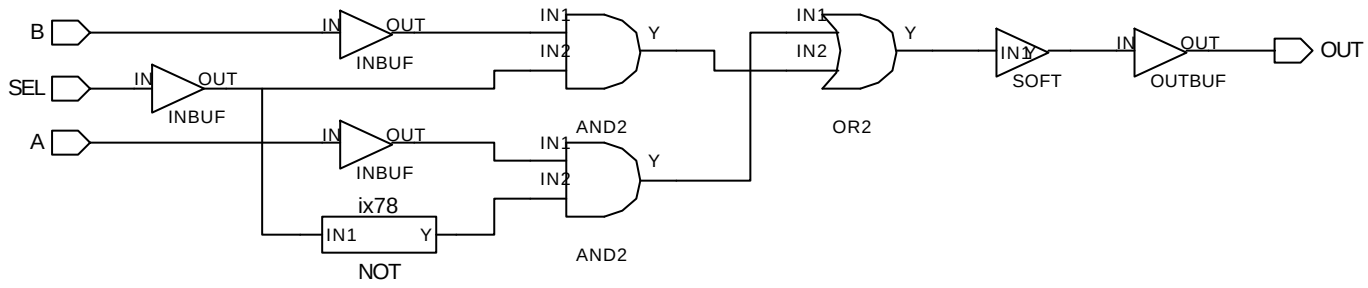

B
SEL
A
IN
OUT
INBUF
IN
OUT
INBUF
IN
OUT
INBUF
IN1
IN2
Y
AND2
IN1
IN2
Y
AND2
ix80
IN1
Y
NOT
IN1
IN2
Y
OR2
IN1
Y
SOFT
IN
OUT
OUTBUF
OUT

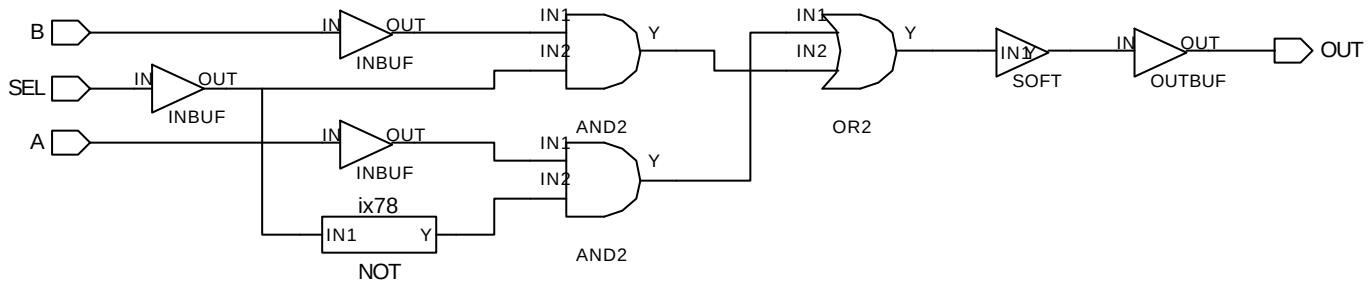

D
C
A
B
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
IN
OUT
INBUF
ix110
ix109
IN1
Y
NOT
AND3
IN2
IN3
OR4
IN4
SOFT
OUTBUF
OUT

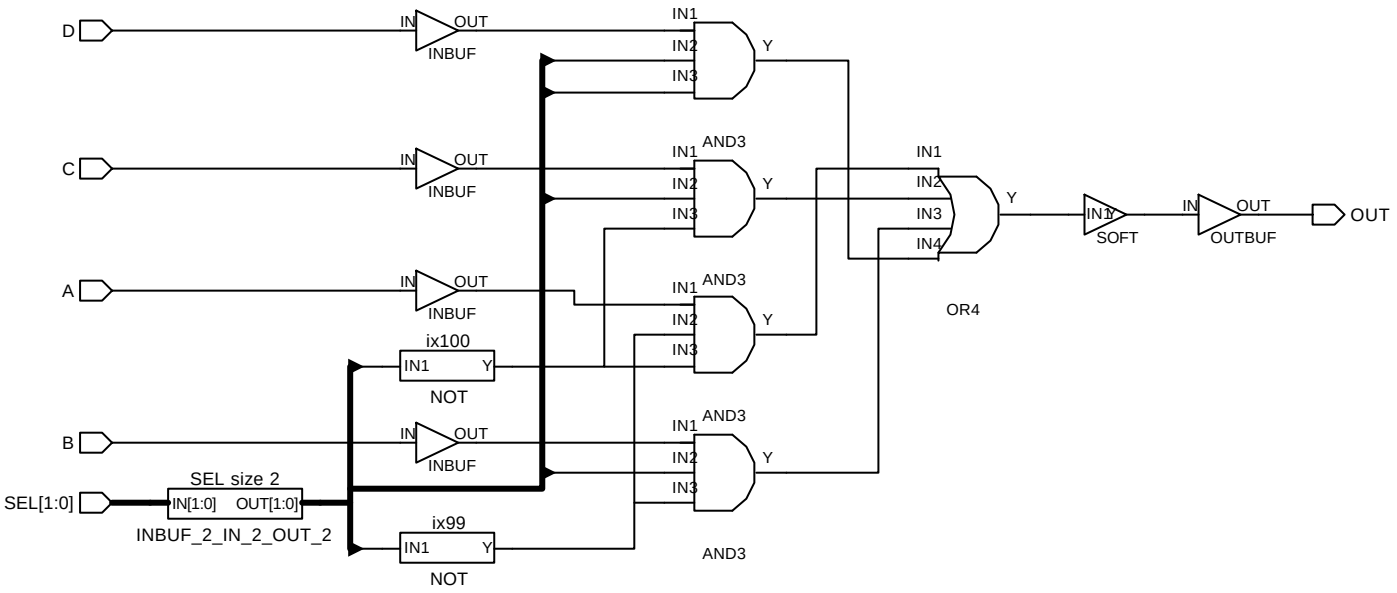

D[1:0]
D size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
B[1:0]
B size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
C[1:0]
C size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
A[1:0]
A size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix148
IN1 Y
NOT
ix149
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
IN1 OR4
IN2
IN3
IN4
Y
OR4
IN1Y
SOFT
IN
OUT
OUTBUF
OUT[1:0]

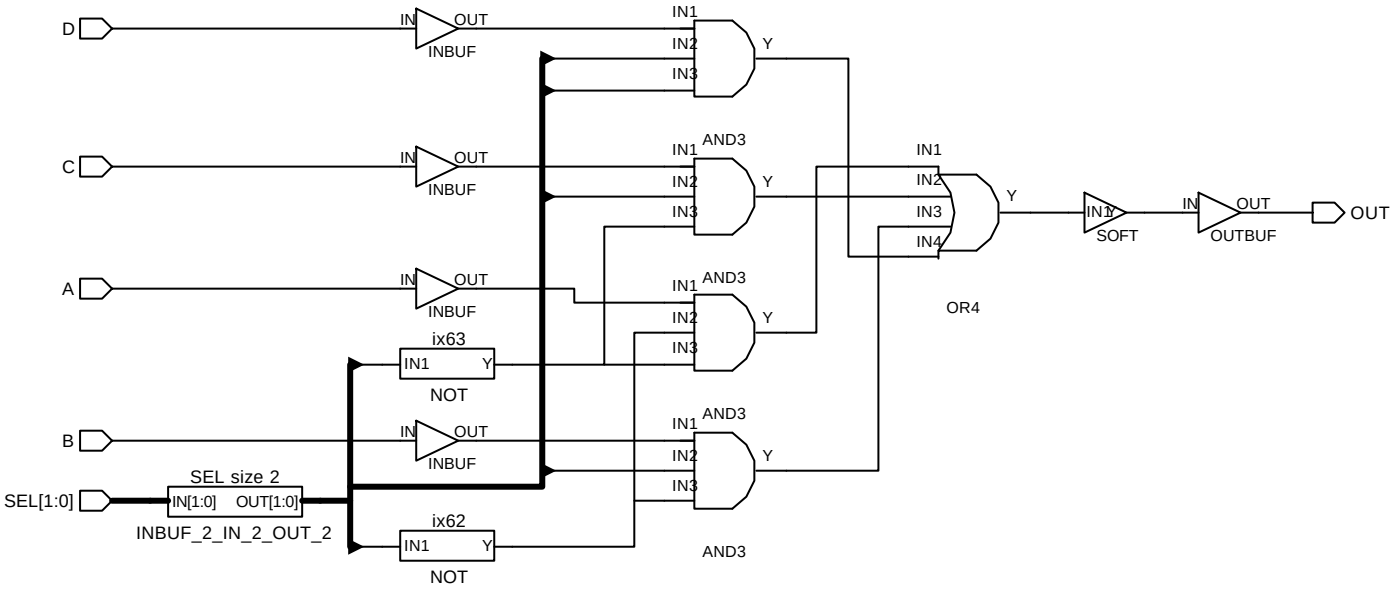

B
SEL
A
IN
OUT
INBUF
IN1
IN2
Y
AND2
OR2
SOFT
OUTBUF
OUT
ix78
NOT

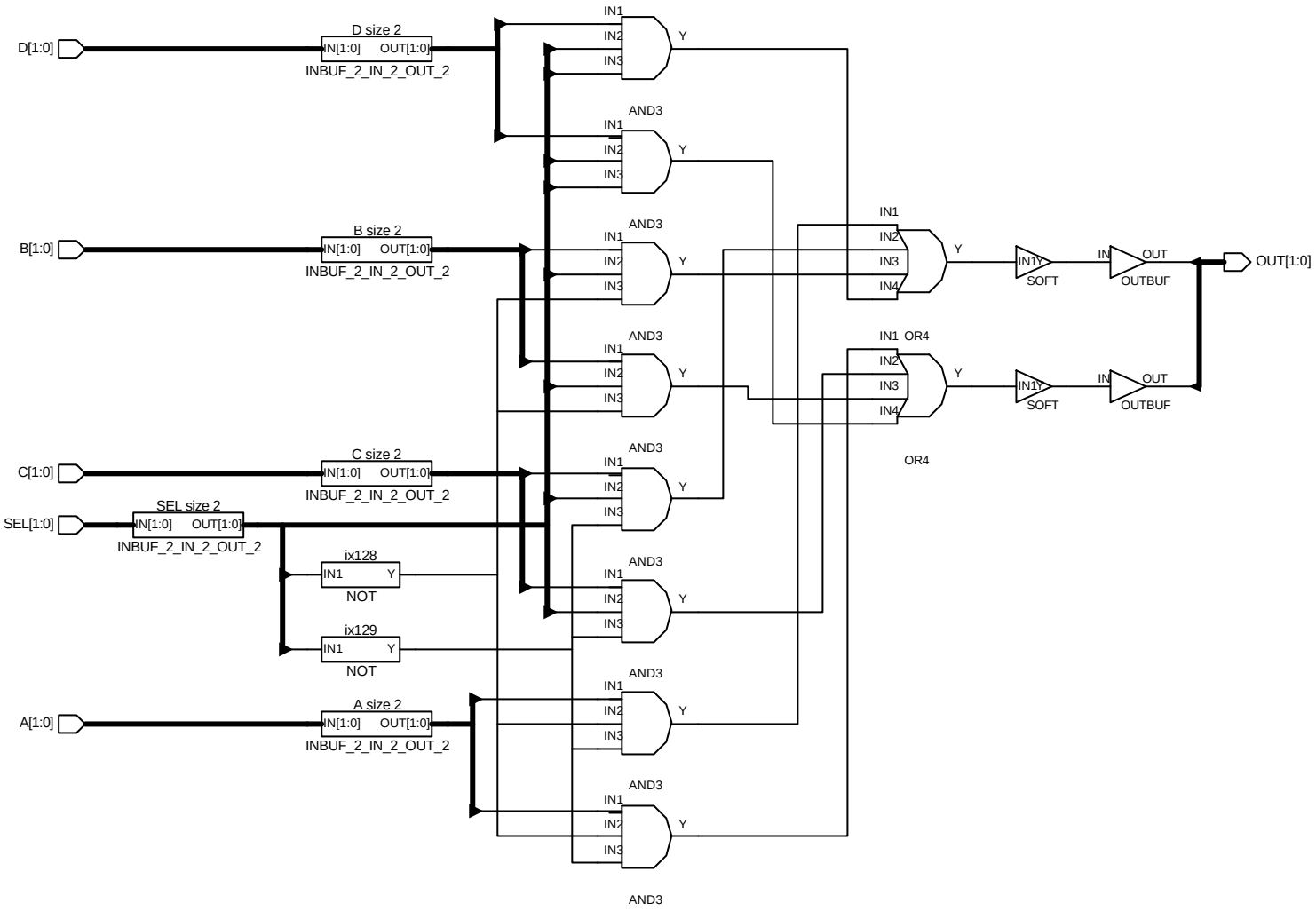

D
C
A
B
SEL[1:0]
IN
OUT
INBUF
SEL size 2
IN[1:0]
OUT[1:0]
INBUF_2_IN_2_OUT_2
ix100
ix99
IN1
Y
NOT
AND3
IN2
IN3
OR4
IN4
SOFT
OUTBUF
OUT

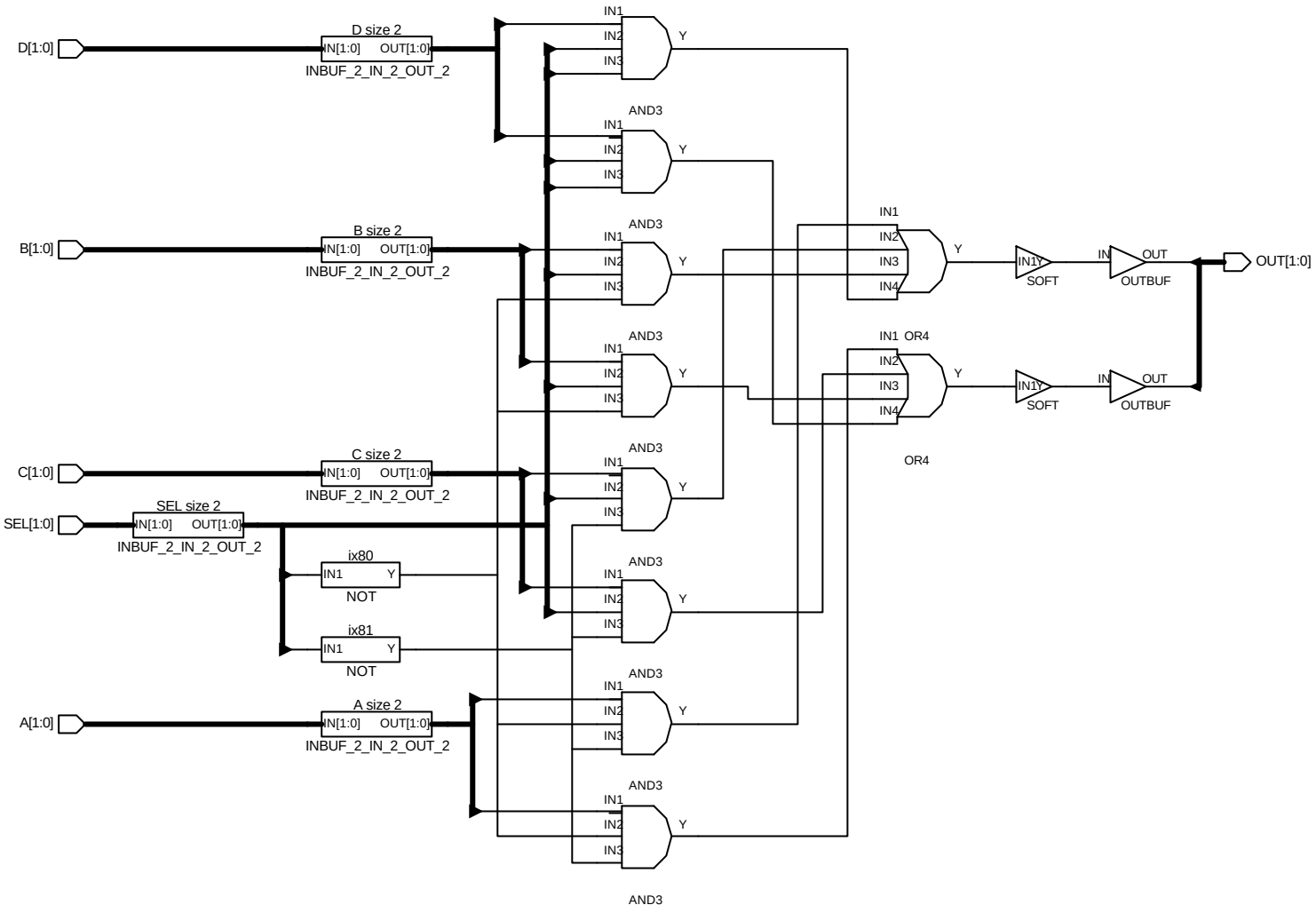

B
SEL
A
IN
OUT
INBUF
IN1
IN2
Y
AND2
OR2
SOFT
OUTBUF
OUT
ix78
NOT

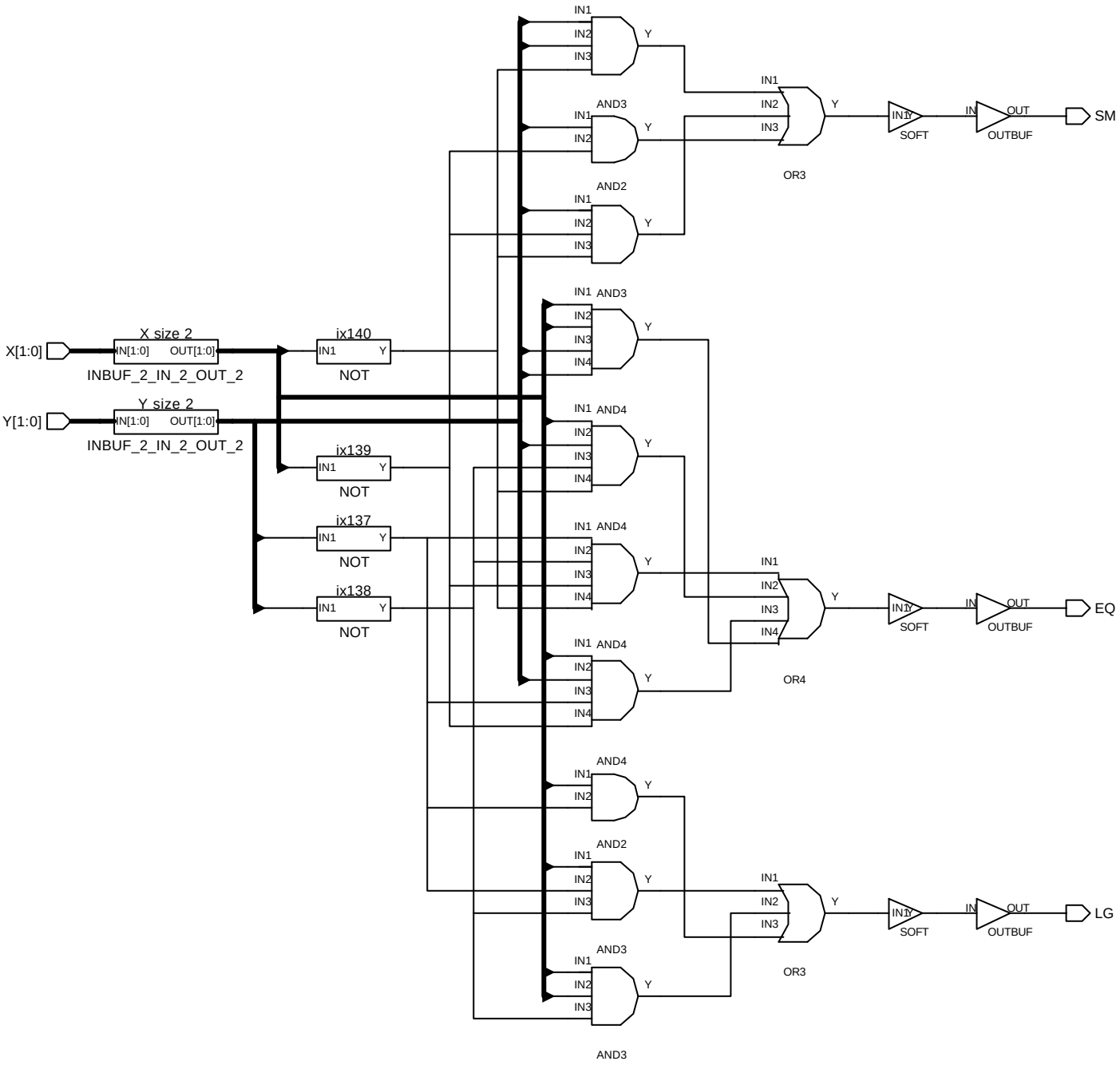

B
SEL
A
IN
OUT
INBUF
IN1
IN2
Y
AND2
OR2
SOFT
OUTBUF
OUT
ix78
NOT

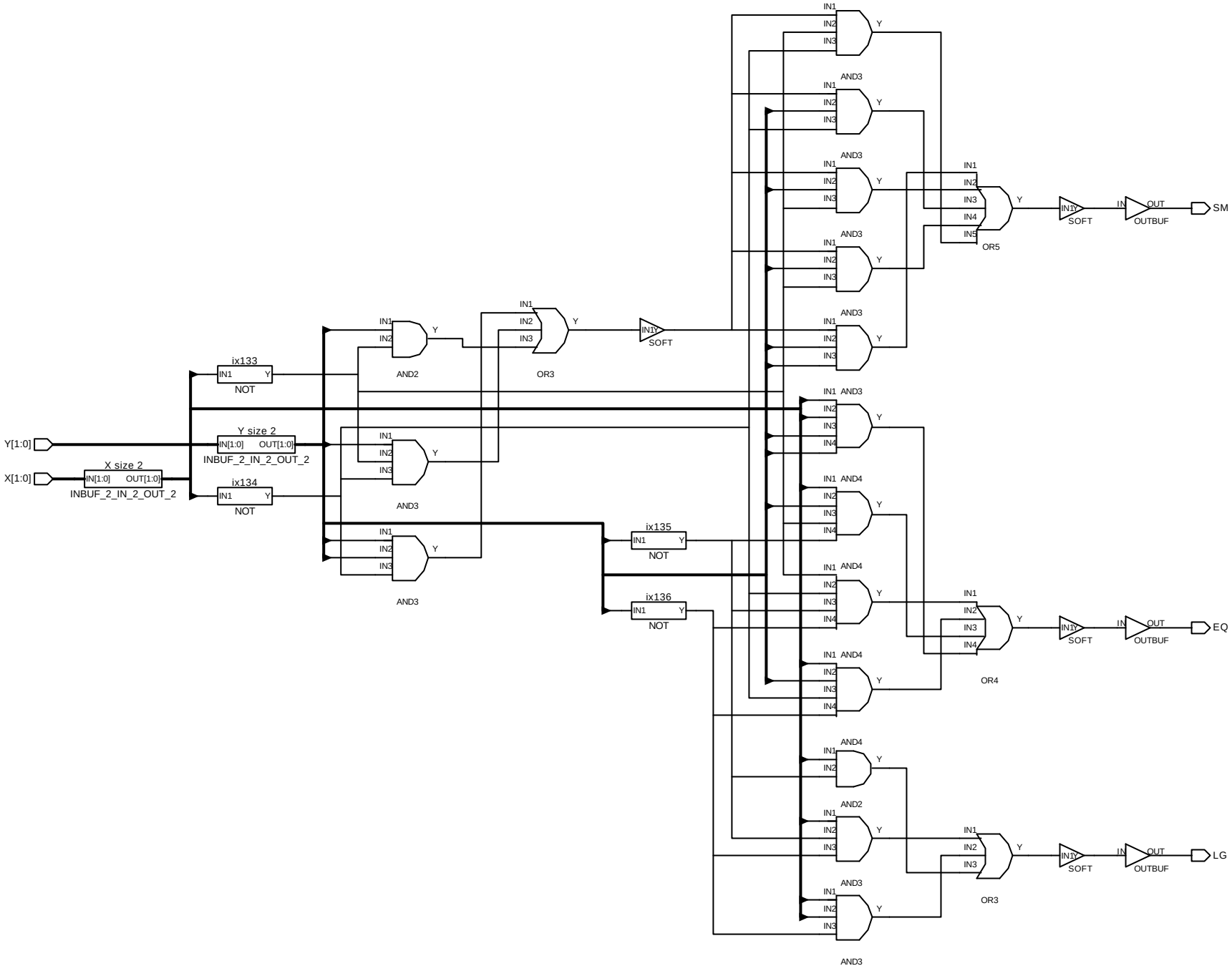

D
C
A
B
SEL[1:0]
IN
OUT
INBUF
SEL size 2
IN[1:0]
OUT[1:0]
INBUF_2_IN_2_OUT_2
ix100
ix99
IN1
Y
NOT
AND3
IN2
IN3
OR4
IN4
SOFT
OUTBUF
OUT

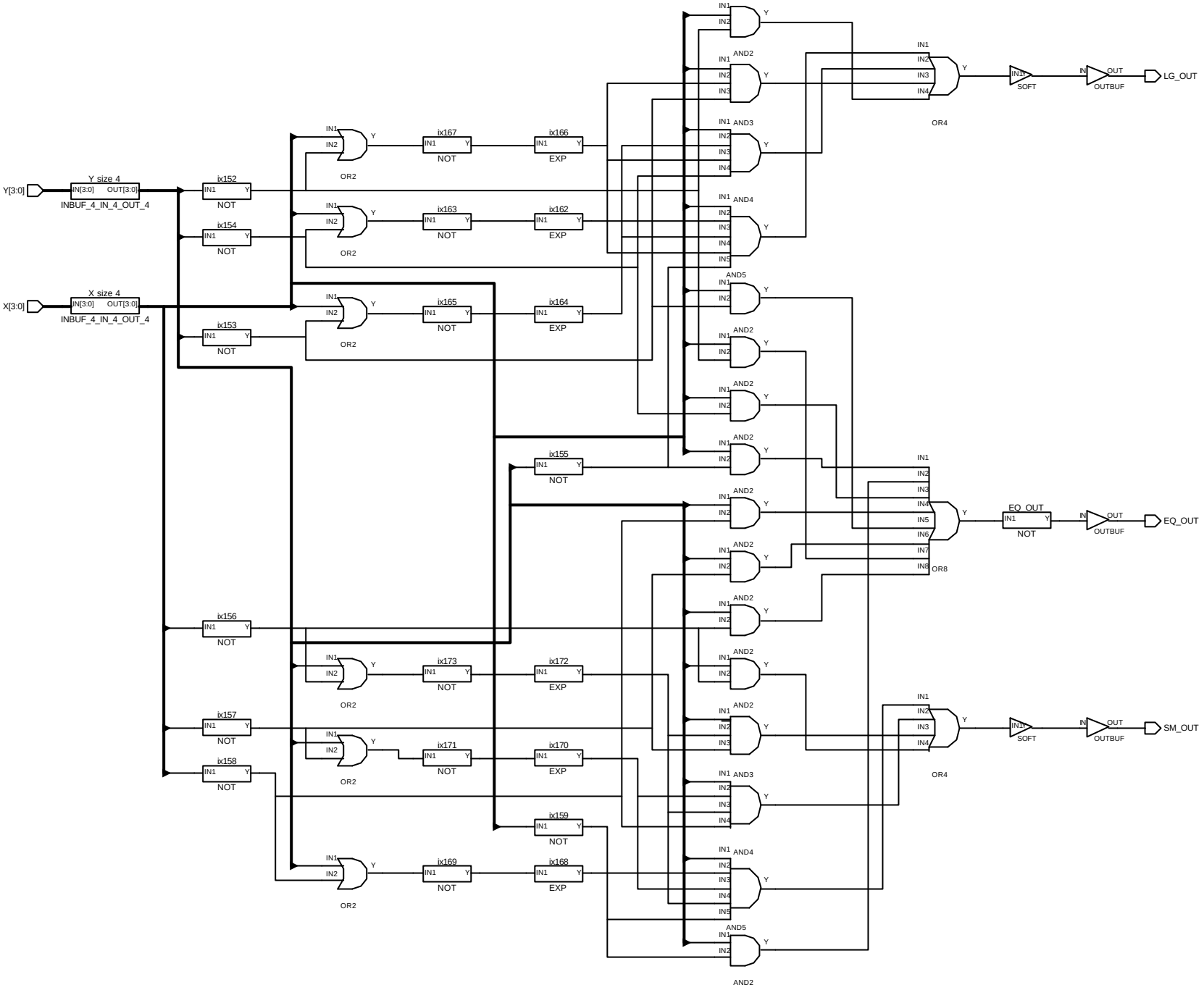

D
C
A
B
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
IN
OUT
INBUF
ix63
ix62
IN1
Y
NOT
AND3
IN2
IN3
OR4
IN4
SOFT
OUTBUF
OUT

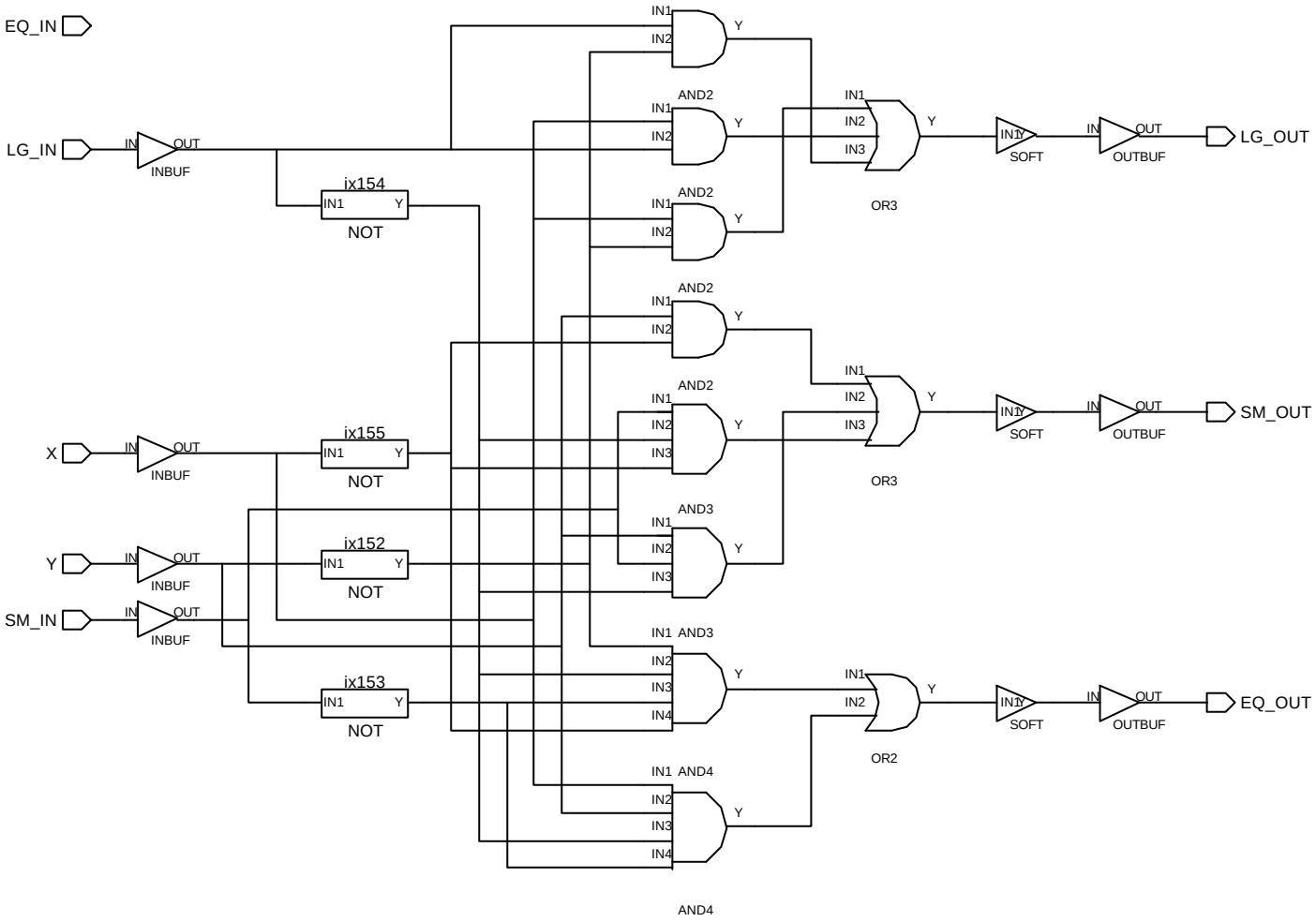

D[1:0]
D size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
B[1:0]
B size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
C[1:0]
C size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
A[1:0]
A size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix128
IN1 Y
NOT
ix129
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
IN1 OR4
IN2
IN3
IN4
Y
OR4
IN1Y
SOFT
IN
OUT
OUTBUF
OUT[1:0]

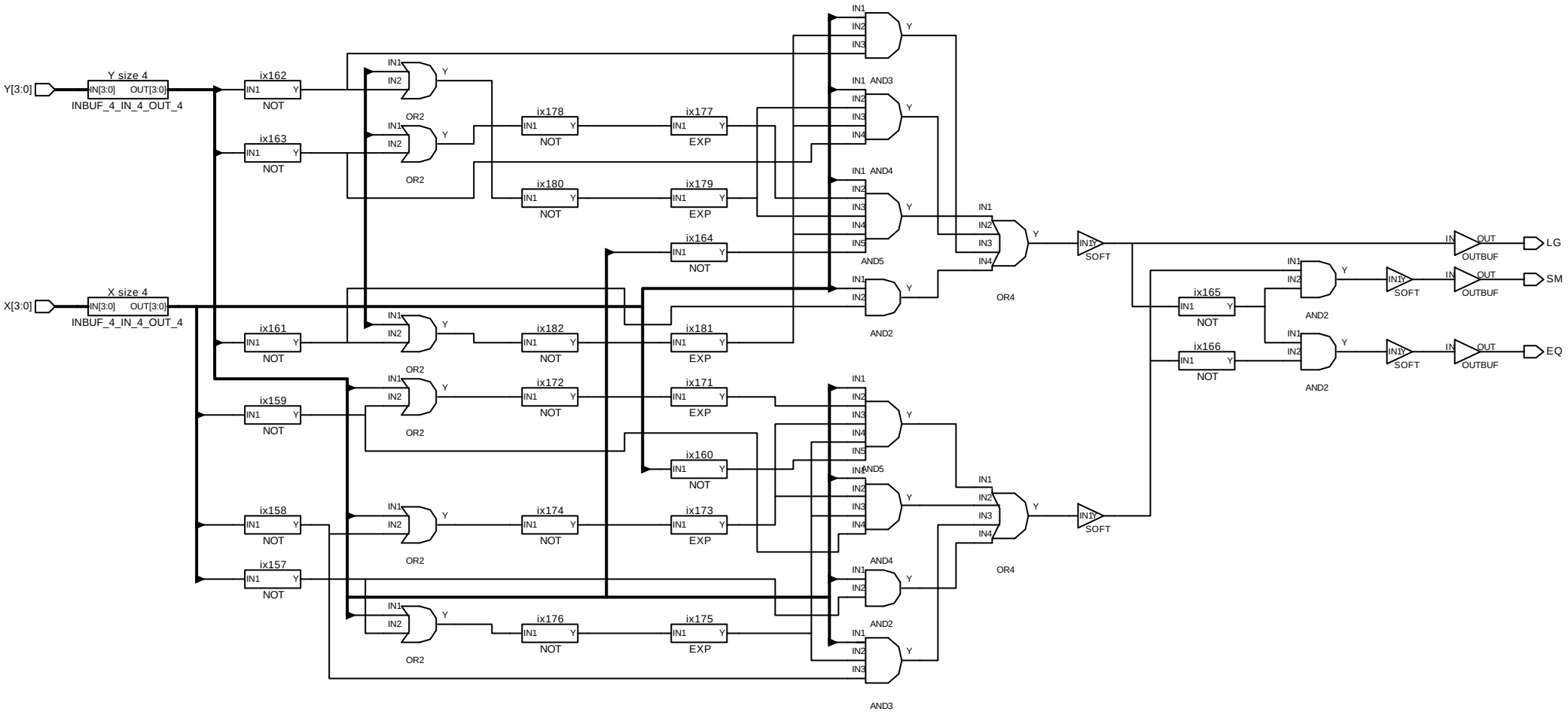

D[1:0]
D size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
B[1:0]
B size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
C[1:0]
C size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
A[1:0]
A size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix80
IN1 Y
NOT
ix81
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
IN1 OR4
IN2
IN3
IN4
Y
OR4
IN1Y
SOFT
IN
OUT
OUTBUF
OUT[1:0]

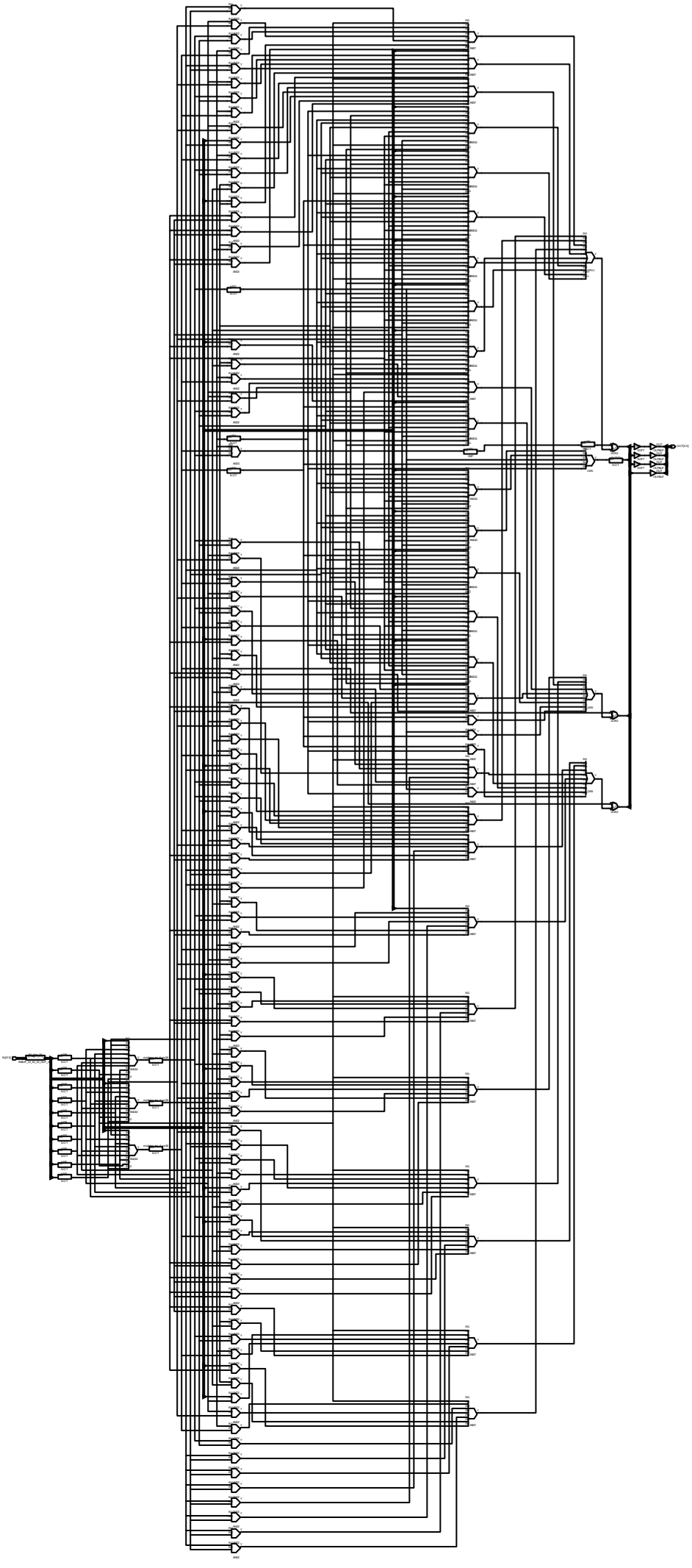

X[1:0]
X size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
Y[1:0]
Y size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix140
IN1 Y
NOT
ix139
IN1 Y
NOT
ix137
IN1 Y
NOT
ix138
IN1 Y
NOT
AND3
AND2
AND4
AND4
AND4
AND2
AND3
AND3
IN1
IN2
IN3
IN4
Y
OR3
OR4
OR3
IN1
IN2
IN3
IN4
Y
IN1 Y
SOFT
IN OUT
OUTBUF
SM
EQ
LG

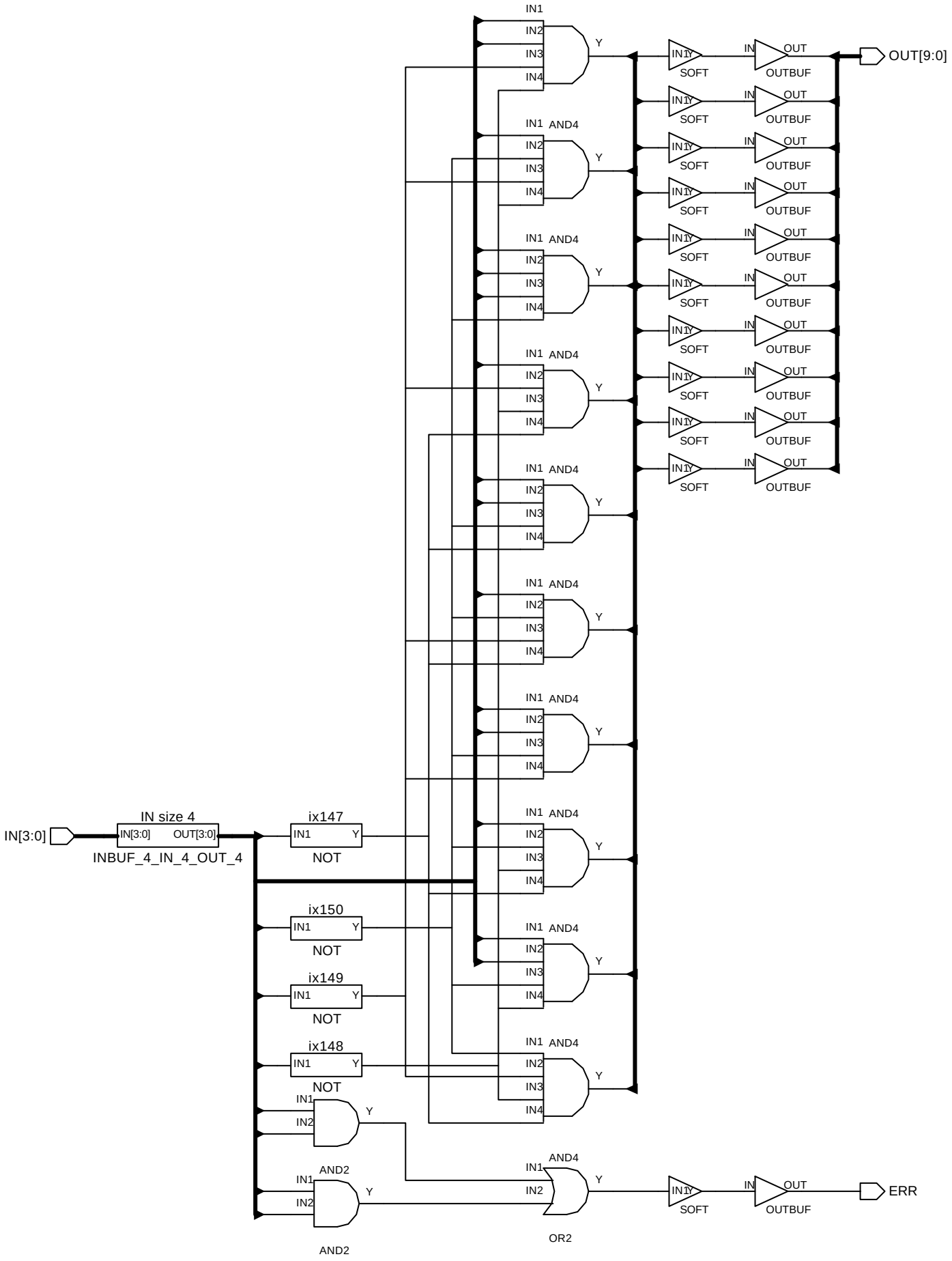

Y[1:0]
X[1:0]
X size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
Y size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix133
IN1 Y
NOT
ix134
IN1 Y
NOT
ix135
IN1 Y
NOT
ix136
IN1 Y
NOT
AND2
AND3
AND3
OR3
SOFT
AND3
AND3
AND3
AND4
AND4
AND4
AND4
AND2
AND3
OR5
OR4
OR3
IN1Y
SOFT
IN OUT
OUTBUF
SM
EQ
LG

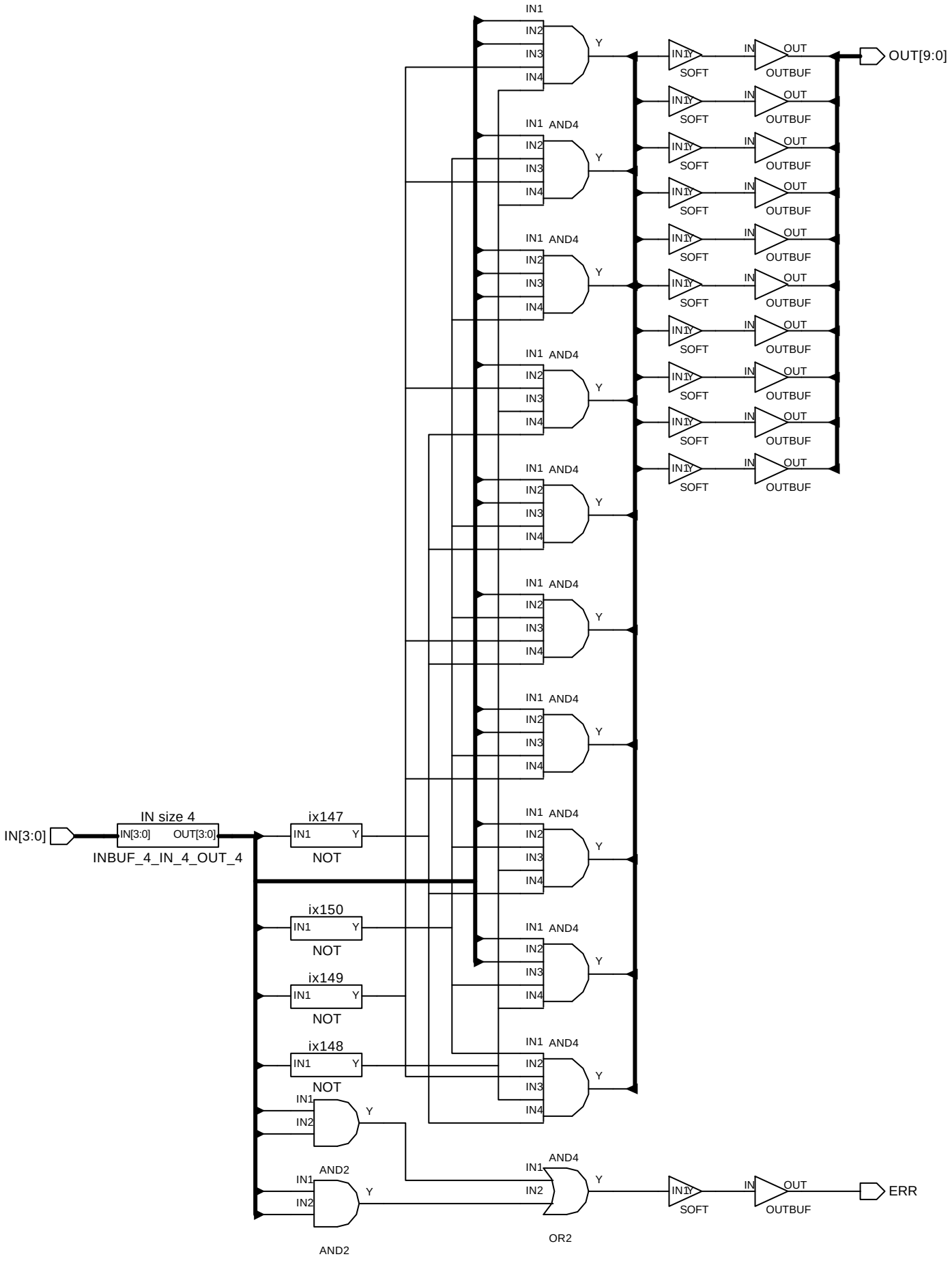

Y[3:0]
Y size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
X[3:0]
X size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
ix152
ix154
ix153
ix156
ix157
ix158
ix155
ix159
NOT
ix167
ix163
ix165
ix173
ix171
ix169
ix166
ix162
ix164
ix172
ix170
ix168
EXP
OR2
AND2
AND3
AND4
AND5
OR4
OR8
EQ_OUT
LG_OUT
SM_OUT
SOFT
OUTBUF

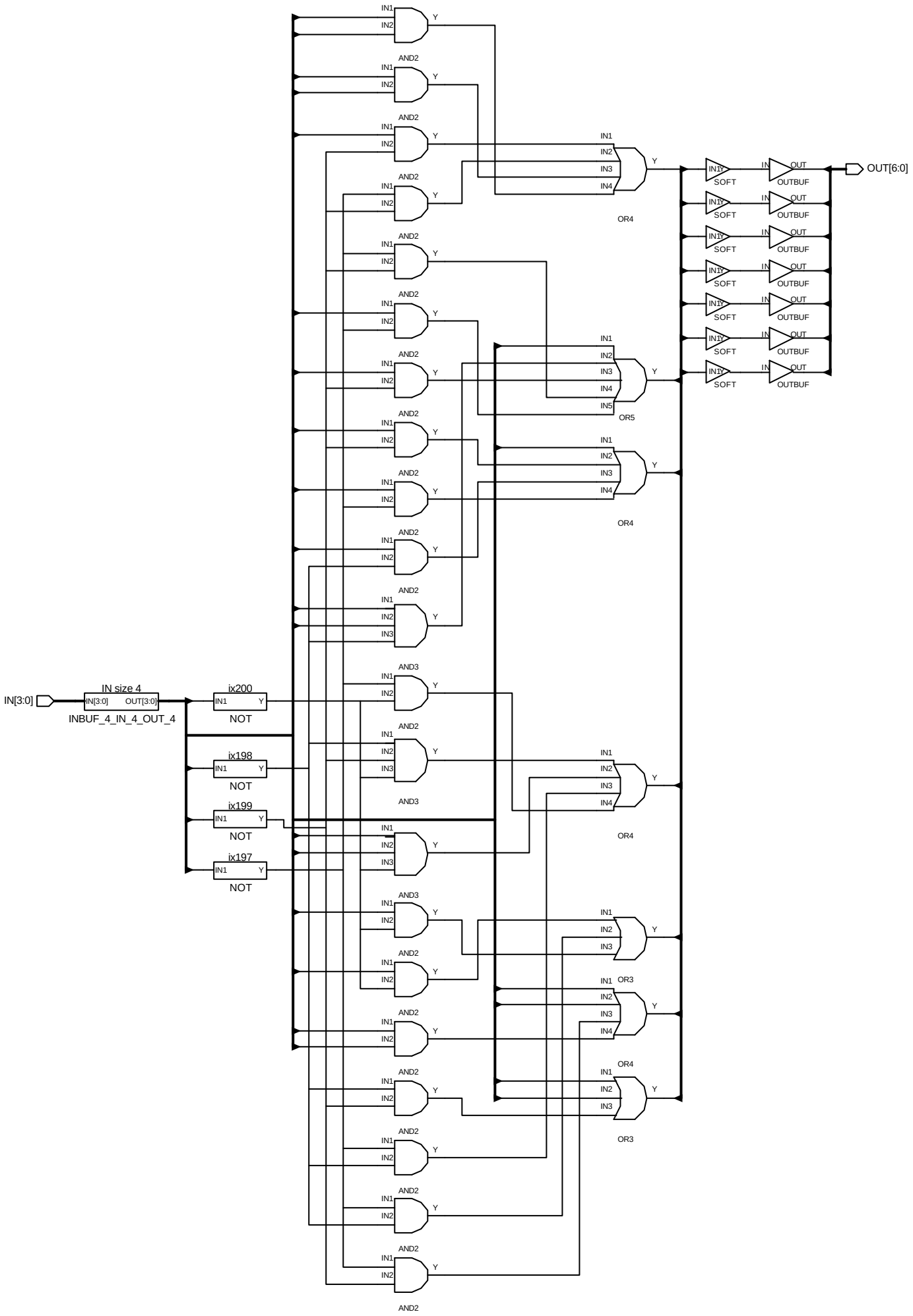

EQ_IN
LG_IN
X
Y
SM_IN
INBUF
OUT
IN
ix154
ix155
ix152
ix153
IN1
Y
NOT
AND2
AND3
AND4
IN2
IN3
IN4
OR3
OR2
SOFT
OUTBUF
LG_OUT
SM_OUT
EQ_OUT

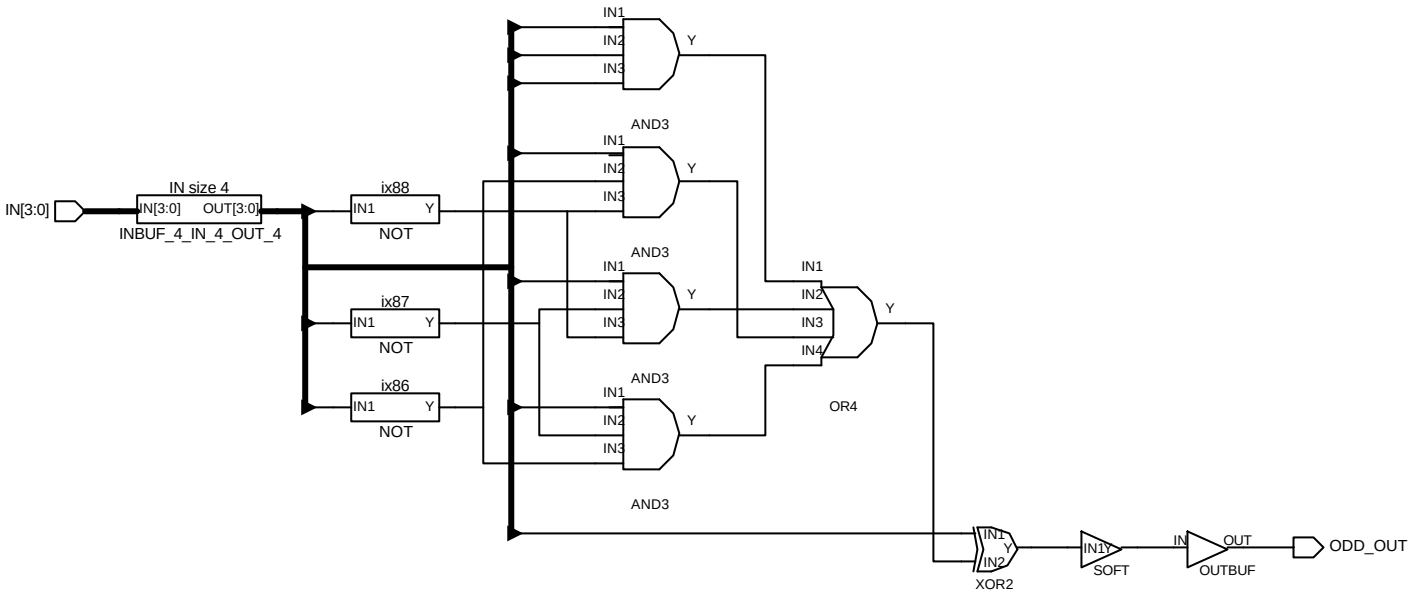

Y[3:0]
Y size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
X[3:0]
X size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
ix162
ix163
ix161
ix159
ix158
ix157
NOT
OR2
ix178
ix180
ix182
ix172
ix174
ix176
ix177
ix179
ix181
ix171
ix173
ix175
EXP
ix164
ix160
AND3
AND4
AND5
AND2
OR4
SOFT
ix165
ix166
OUTBUF
LG
SM
EQ
IN1
IN2
IN3
IN4
IN5
Y

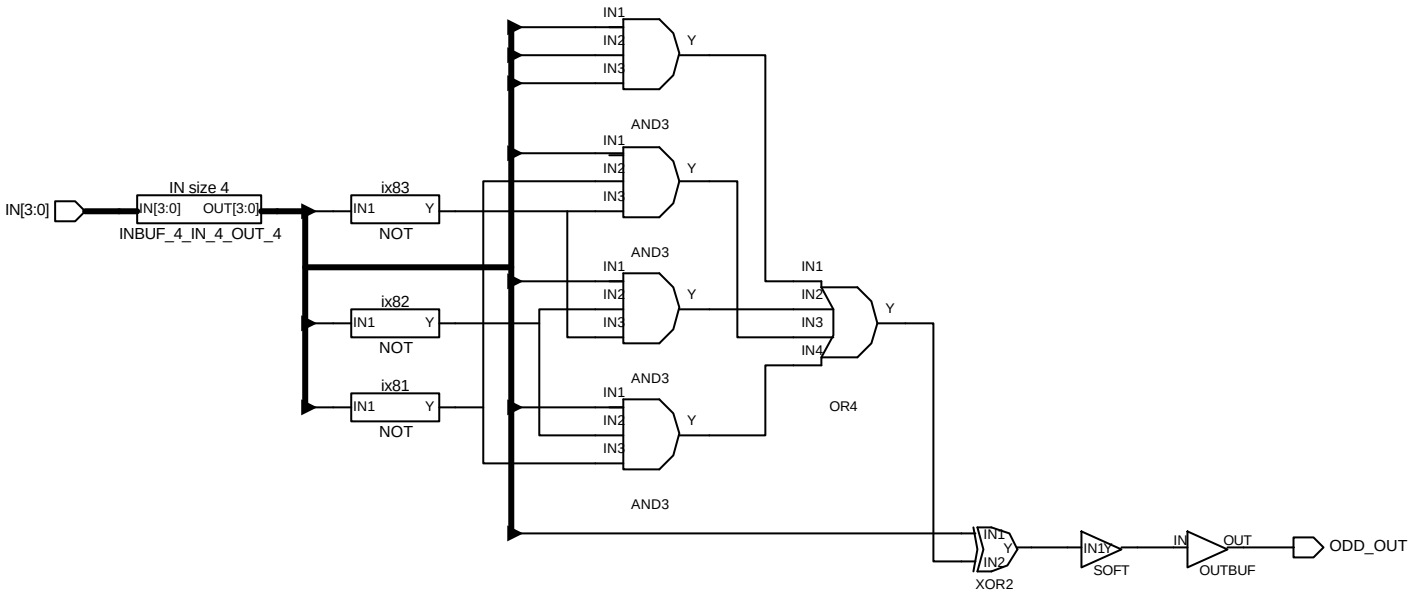


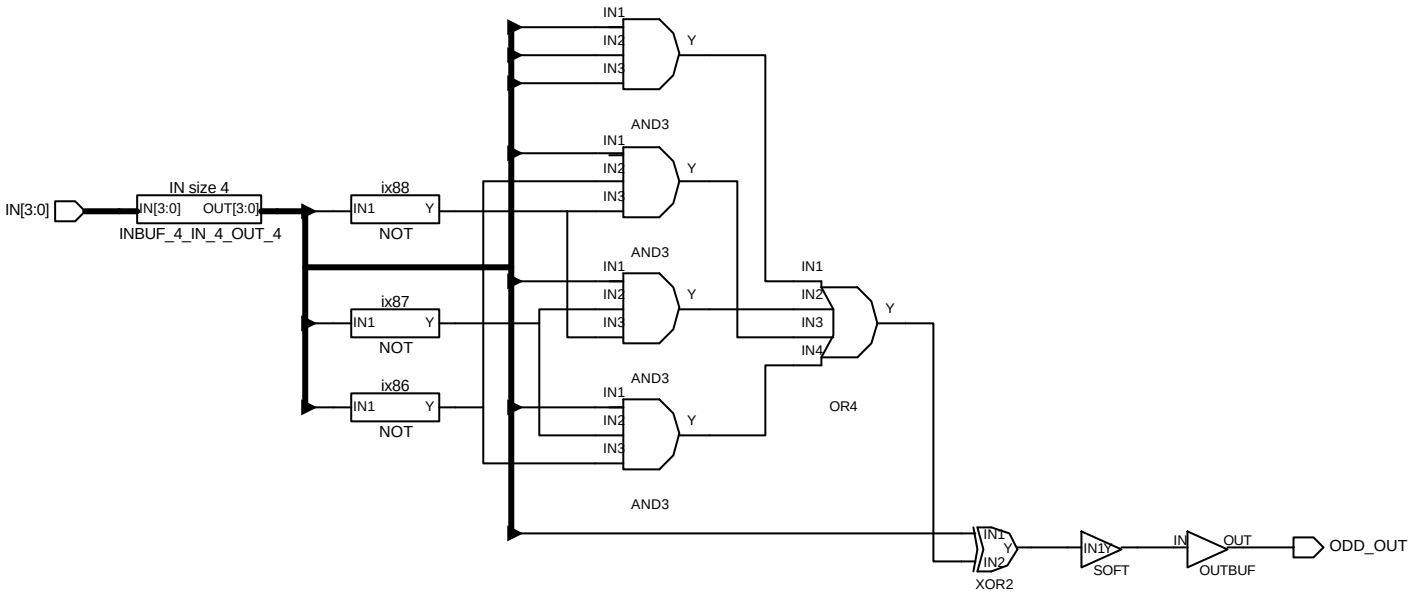

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix147
IN1
Y
NOT
ix150
NOT
ix149
NOT
ix148
NOT
AND4
IN1
IN2
IN3
IN4
Y
AND2
OR2
SOFT
IN1Y
OUTBUF
IN
OUT
OUT[9:0]
ERR

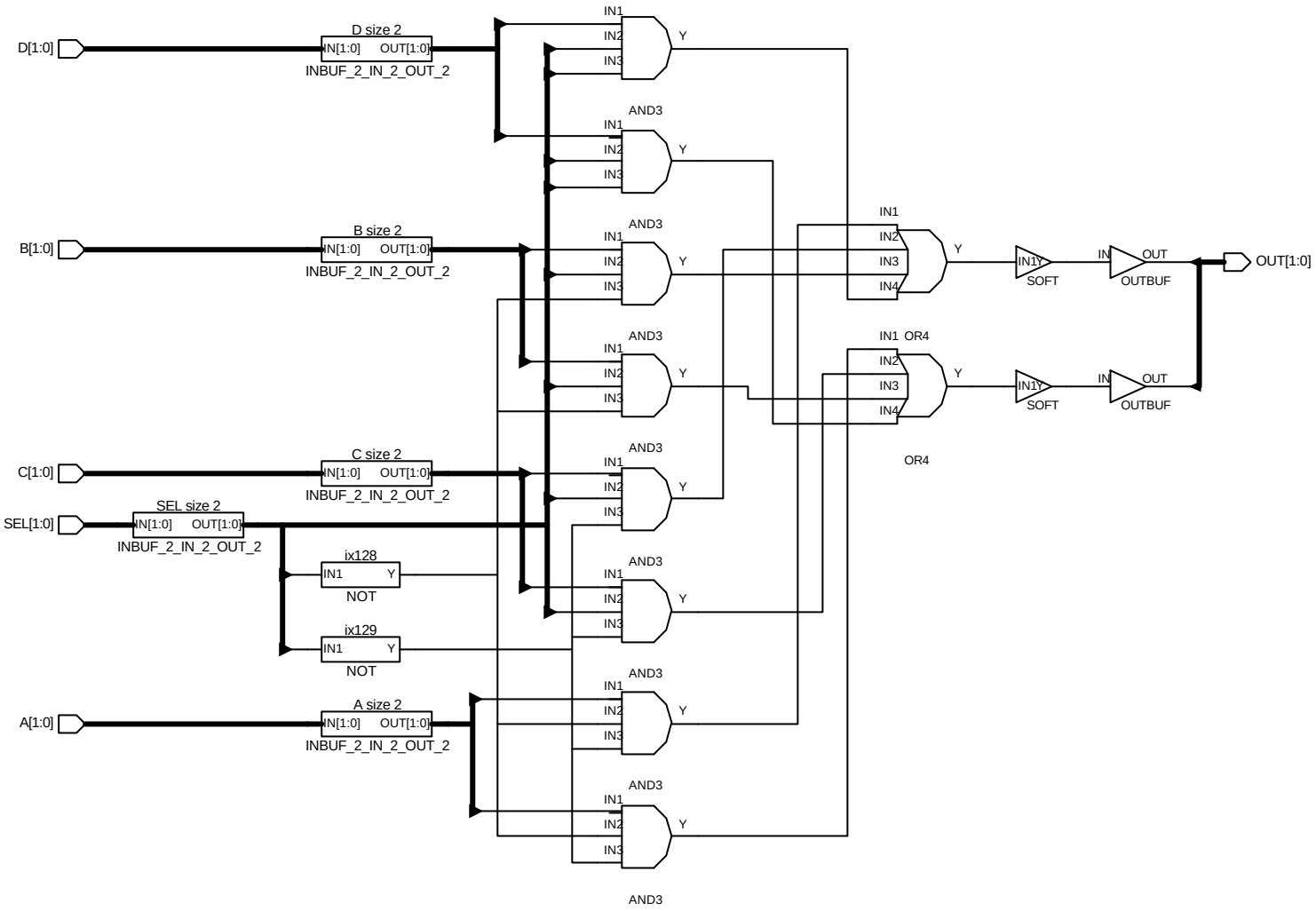

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix147
IN1
Y
NOT
ix150
NOT
ix149
NOT
ix148
NOT
AND4
IN1
IN2
IN3
IN4
Y
AND2
OR2
SOFT
IN1Y
OUTBUF
IN
OUT
OUT[9:0]
ERR

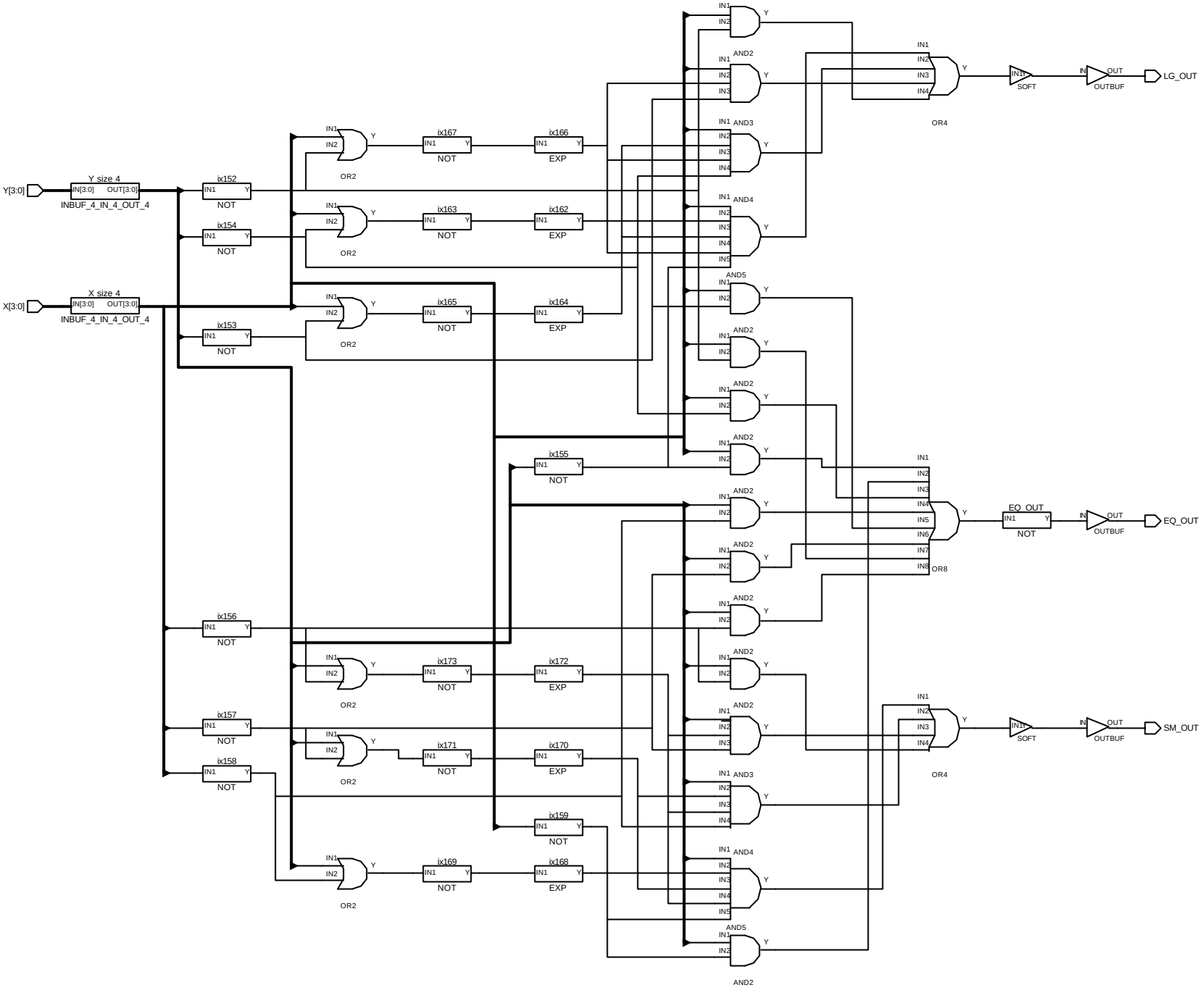

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix200
NOT
ix198
NOT
ix199
NOT
ix197
NOT
AND2
AND3
OR4
OR5
OR3
SOFT
OUTBUF
OUT[6:0]

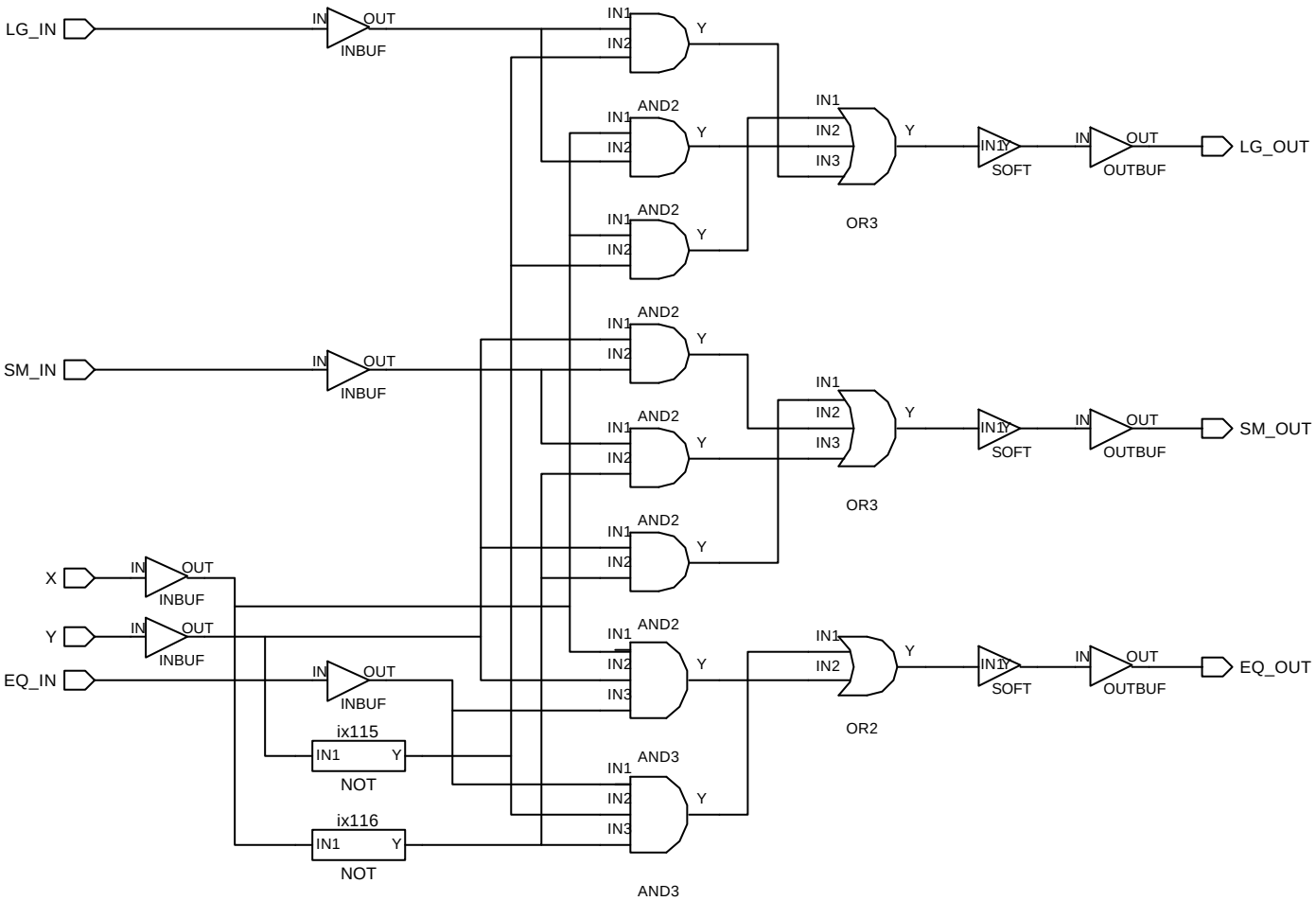

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix88
IN1
Y
NOT
ix87
IN1
Y
NOT
ix86
IN1
Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
OR4
IN1
IN2
IN3
IN4
Y
XOR2
IN1
IN2
Y
SOFT
IN1Y
OUTBUF
IN
OUT
ODD_OUT

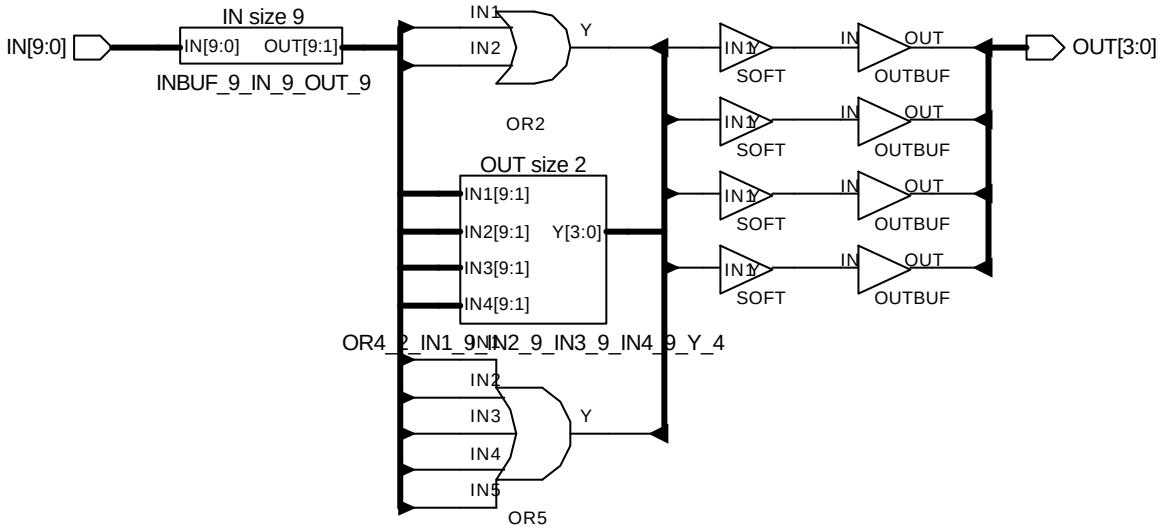

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix83
IN1
Y
NOT
ix82
IN1
Y
NOT
ix81
IN1
Y
NOT
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
IN4
Y
OR4
IN1
Y
IN2
XOR2
IN1Y
SOFT
IN
OUT
OUTBUF
ODD_OUT

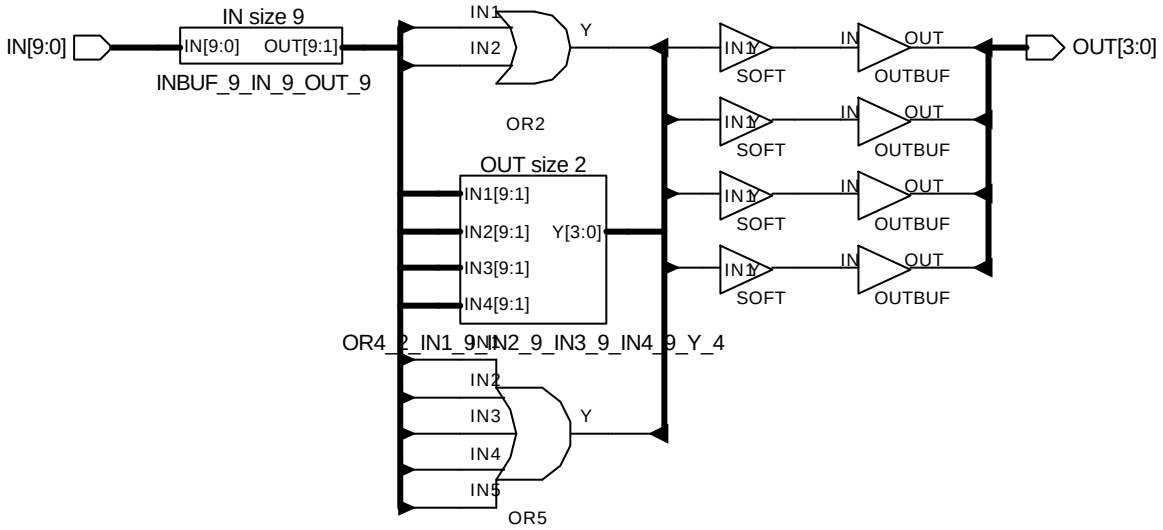

IN[3:0]
IN size 4
IN[3:0]
OUT[3:0]
INBUF_4_IN_4_OUT_4
ix88
IN1
Y
NOT
ix87
IN1
Y
NOT
ix86
IN1
Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
AND3
IN1
IN2
IN3
Y
OR4
IN1
IN2
IN3
IN4
Y
XOR2
IN1
IN2
Y
SOFT
IN1Y
OUTBUF
IN
OUT
ODD_OUT

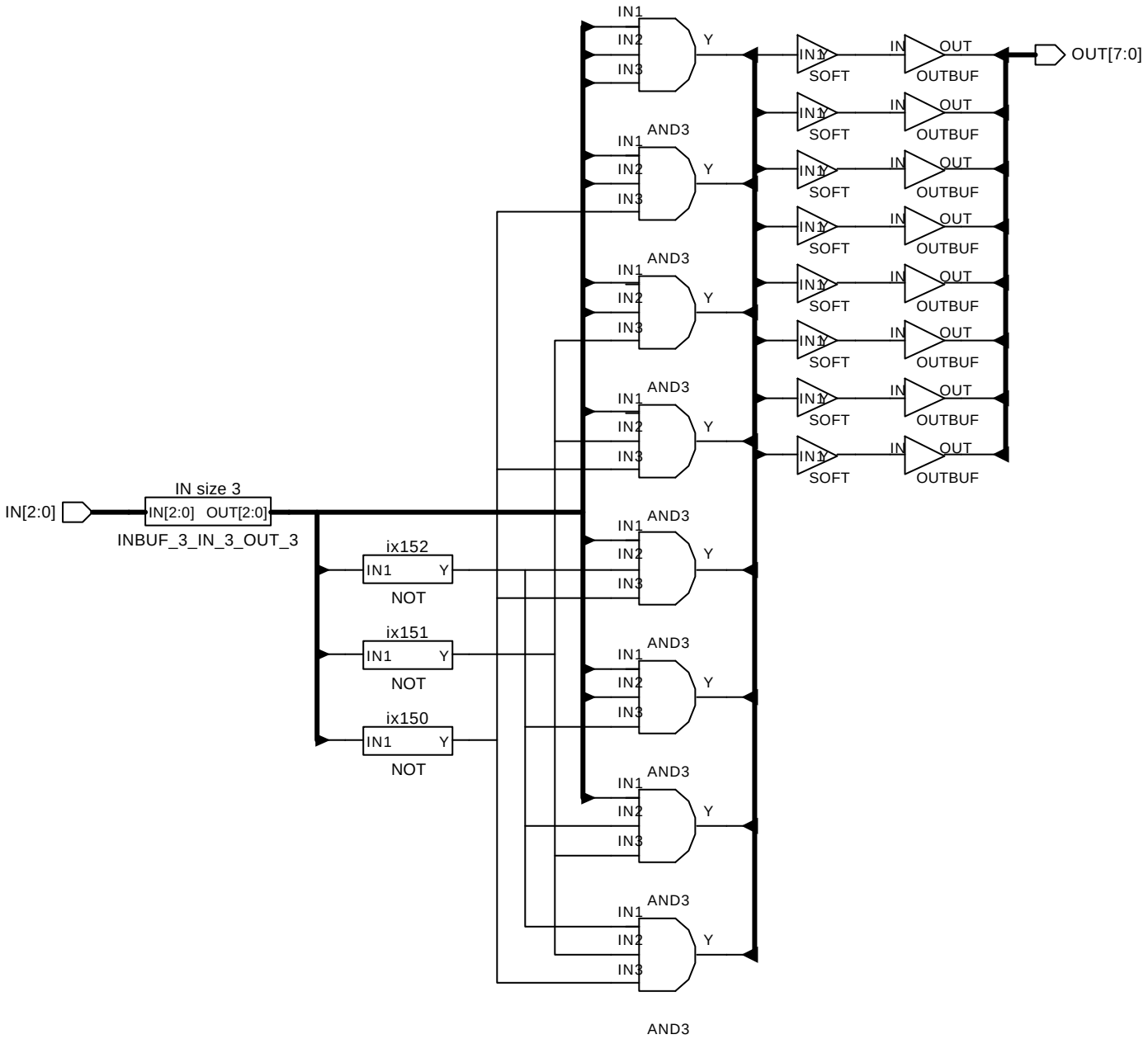

D[1:0]
D size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
B[1:0]
B size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
C[1:0]
C size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
SEL[1:0]
SEL size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
A[1:0]
A size 2
IN[1:0] OUT[1:0]
INBUF_2_IN_2_OUT_2
ix128
IN1 Y
NOT
ix129
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
IN1 OR4
IN2
IN3
IN4
Y
OR4
IN1Y
SOFT
IN
OUT
OUTBUF
OUT[1:0]

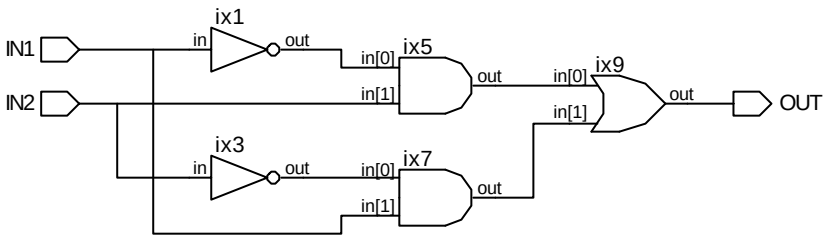

Y[3:0]
Y size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
X[3:0]
X size 4
IN[3:0] OUT[3:0]
INBUF_4_IN_4_OUT_4
ix152
ix154
ix153
ix156
ix157
ix158
ix155
ix159
NOT
ix167
ix163
ix165
ix173
ix171
ix169
ix166
ix162
ix164
ix172
ix170
ix168
EXP
OR2
AND2
AND3
AND4
AND5
OR4
OR8
EQ_OUT
LG_OUT
SM_OUT
SOFT
OUTBUF

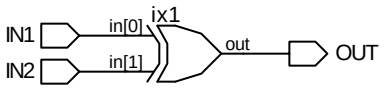

LG_IN
IN
OUT
INBUF
IN1
IN2
Y
AND2
IN3
OR3
IN1Y
SOFT
OUTBUF
LG_OUT
SM_IN
SM_OUT
X
Y
EQ_IN
ix115
NOT
ix116
AND3
OR2
EQ_OUT

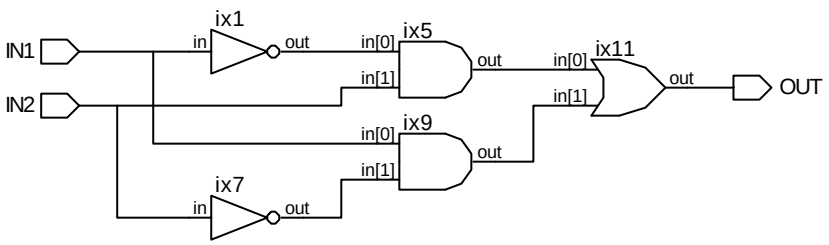

IN[9:0]
IN size 9
IN[9:0]
OUT[9:1]
INBUF_9_IN_9_OUT_9
IN1
IN2
Y
OR2
OUT size 2
IN1[9:1]
IN2[9:1]
IN3[9:1]
IN4[9:1]
Y[3:0]
OR4_2_IN1_9_IN2_9_IN3_9_IN4_9_Y_4
IN1
IN2
IN3
IN4
IN5
Y
OR5
IN
Y
SOFT
IN
OUT
OUTBUF
OUT[3:0]

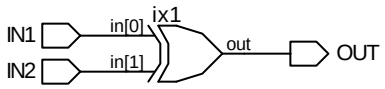

IN[9:0]
IN size 9
IN[9:0]
OUT[9:1]
INBUF_9_IN_9_OUT_9
IN1
IN2
Y
OR2
OUT size 2
IN1[9:1]
IN2[9:1]
IN3[9:1]
IN4[9:1]
Y[3:0]
OR4_2_IN1_9_IN2_9_IN3_9_IN4_9_Y_4
IN1
IN2
IN3
IN4
IN5
Y
OR5
IN
Y
SOFT
IN
OUT
OUTBUF
OUT[3:0]

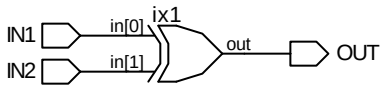

IN[2:0]
IN size 3
IN[2:0] OUT[2:0]
INBUF_3_IN_3_OUT_3
ix152
IN1 Y
NOT
ix151
IN1 Y
NOT
ix150
IN1 Y
NOT
AND3
IN1
IN2
IN3
Y
AND3
AND3
AND3
AND3
AND3
AND3
AND3
IN
Y
SOFT
IN
OUT
OUTBUF
OUT[7:0]

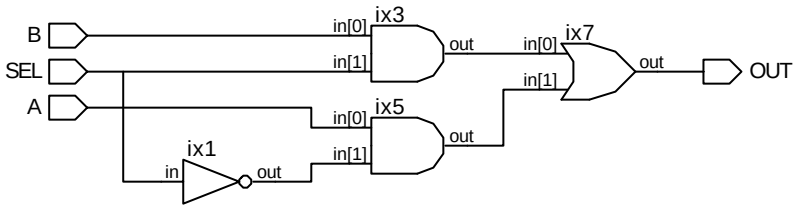

IN1
IN2
ix1
in
out
ix3
in
out
ix5
in[0]
in[1]
out
ix7
in[0]
in[1]
out
ix9
in[0]
in[1]
out
OUT

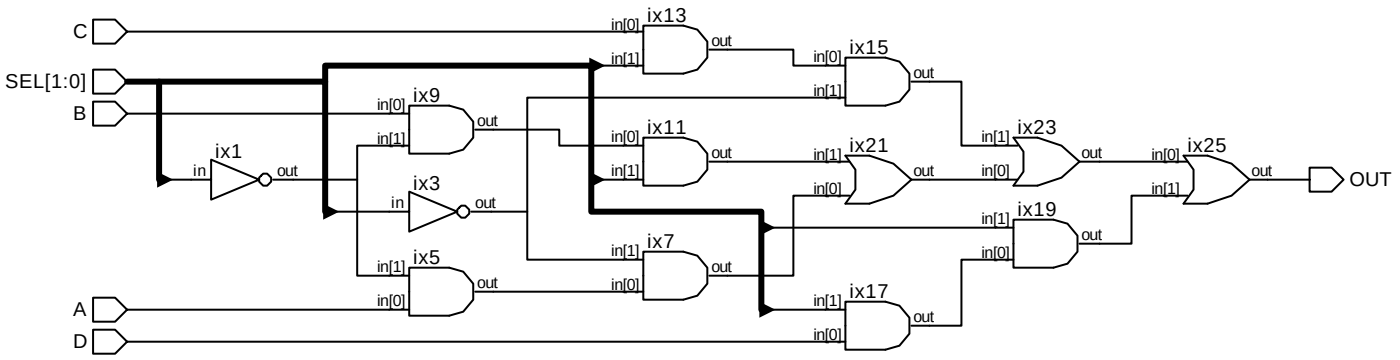

IN1
IN2
in[0]
in[1]
ix1
out
OUT

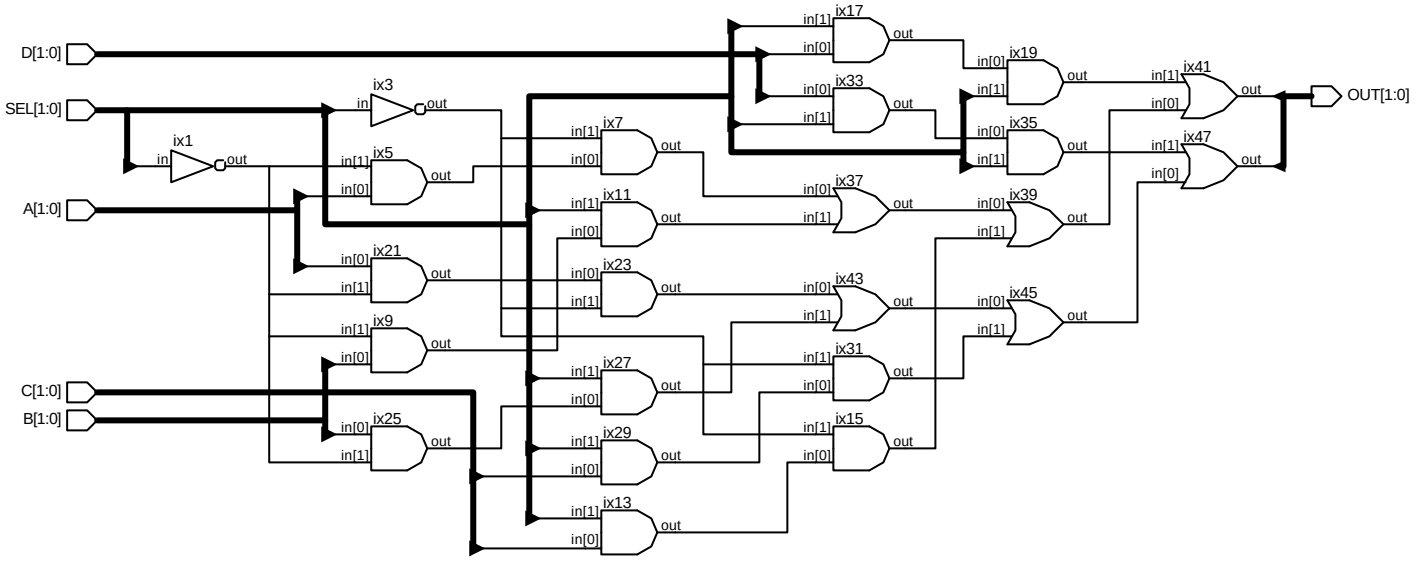

IN1
IN2
ix1
in
out
ix5
in[0]
in[1]
out
ix11
in[0]
in[1]
out
OUT
ix9
in[0]
in[1]
out
ix7
in
out

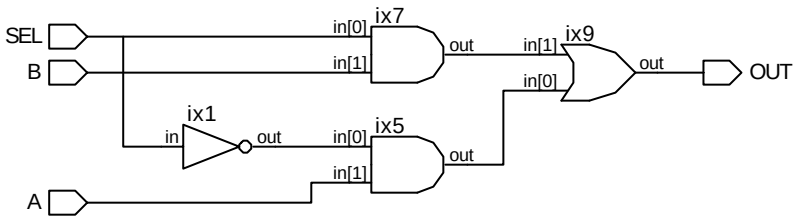

IN1
IN2
in[0]
in[1]
ix1
out
OUT

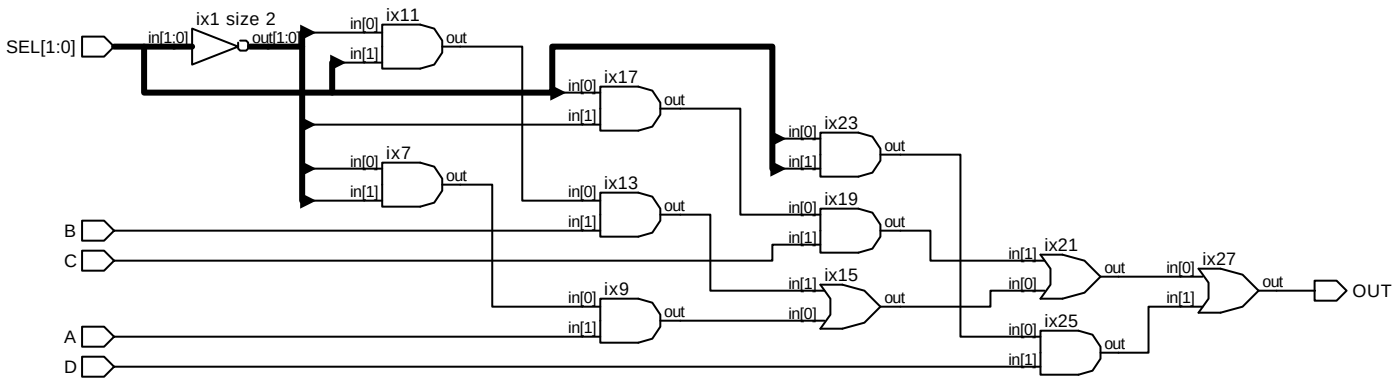

IN1
IN2
in[0]
in[1]
ix1
out
OUT

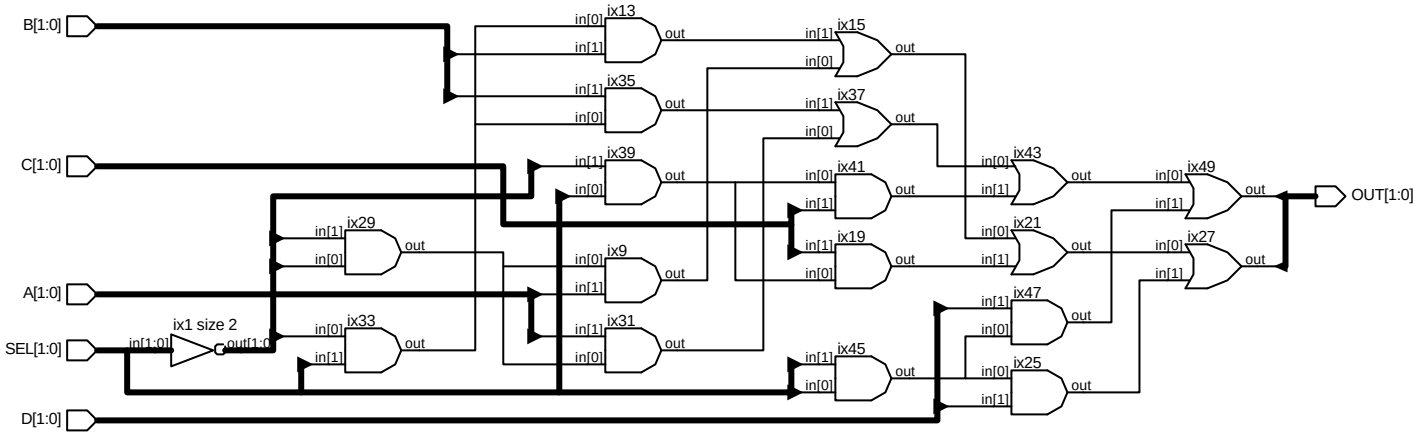

B
SEL
A
ix3
in[0]
in[1]
out
ix7
in[0]
in[1]
out
OUT
ix5
in[0]
in[1]
out
ix1
in
out

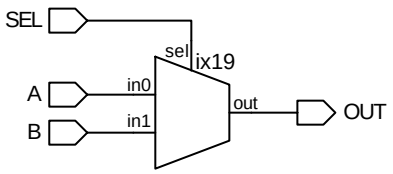

C
SEL[1:0]
B
A
D
ix1
in
out
ix3
in
out
ix5
in[1]
in[0]
out
ix7
in[1]
in[0]
out
ix9
in[0]
in[1]
out
ix11
in[0]
in[1]
out
ix13
in[0]
in[1]
out
ix15
in[0]
in[1]
out
ix17
in[1]
in[0]
out
ix19
in[1]
in[0]
out
ix21
in[1]
in[0]
out
ix23
in[1]
in[0]
out
ix25
in[0]
in[1]
out
OUT

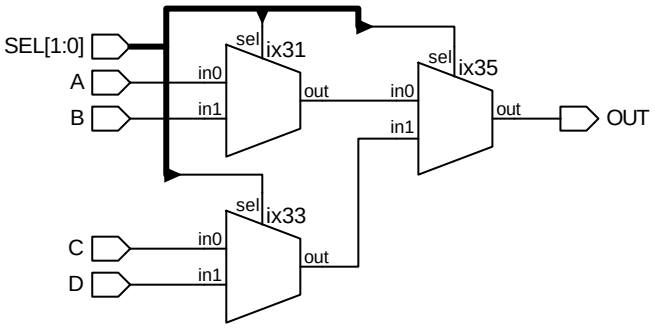

D[1:0]
SEL[1:0]
A[1:0]
C[1:0]
B[1:0]
OUT[1:0]
ix1
ix3
ix5
ix7
ix9
ix11
ix13
ix15
ix17
ix19
ix21
ix23
ix25
ix27
ix29
ix31
ix33
ix35
ix37
ix39
ix41
ix43
ix45
ix47
in
in[0]
in[1]
out

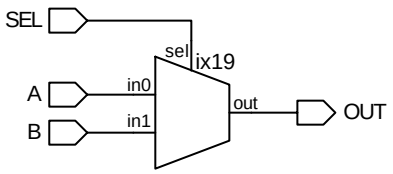

SEL
B
A
ix7
in[0]
in[1]
out
ix9
in[1]
in[0]
out
OUT
ix1
in
out
ix5
in[0]
in[1]
out

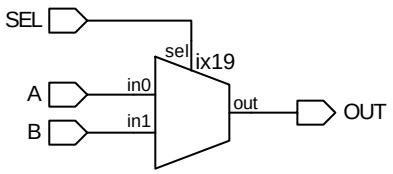

SEL[1:0]
ix1 size 2
in[1:0]
out[1:0]
ix11
in[0]
in[1]
out
ix17
ix7
ix13
ix9
ix23
ix19
ix15
ix21
ix25
ix27
B
C
A
D
OUT

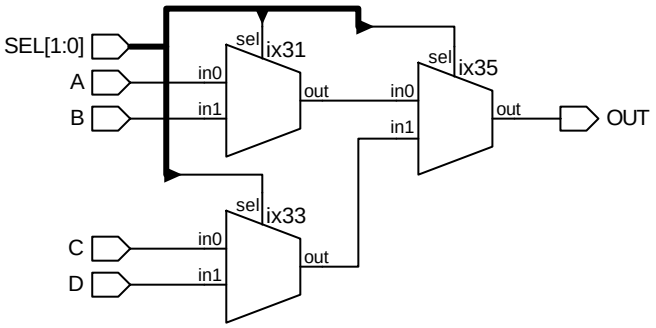

B[1:0]
C[1:0]
A[1:0]
SEL[1:0]
D[1:0]
ix1 size 2
in[1:0]
out[1:0]
ix13
ix35
ix39
ix29
ix9
ix33
ix31
ix15
ix37
ix41
ix19
ix45
ix43
ix21
ix47
ix25
ix49
ix27
in[0]
in[1]
out
OUT[1:0]

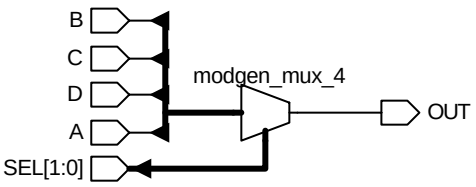

SEL
sel
ix19
A
in0
B
in1
out
OUT

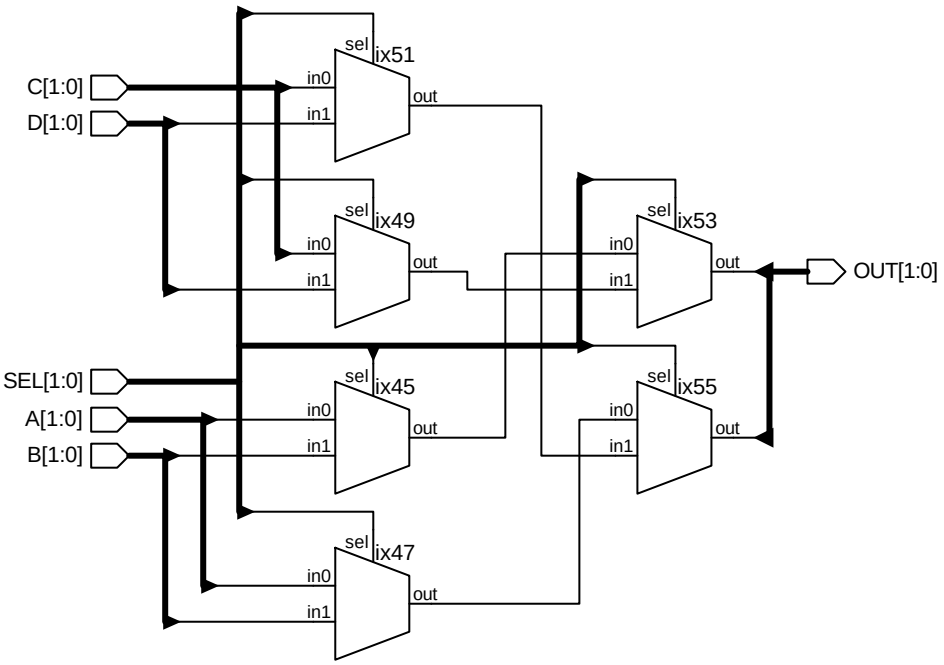

SEL[1:0]
A
B
C
D
sel
ix31
in0
in1
out
sel
ix33
in0
in1
out
sel
ix35
in0
in1
out
OUT

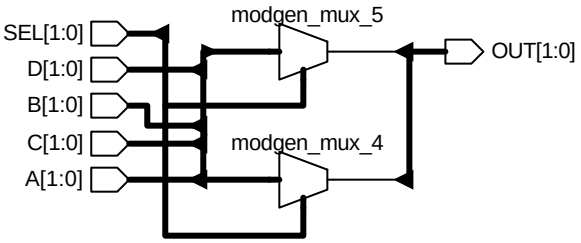

SEL
sel
ix19
A
in0
B
in1
out
OUT

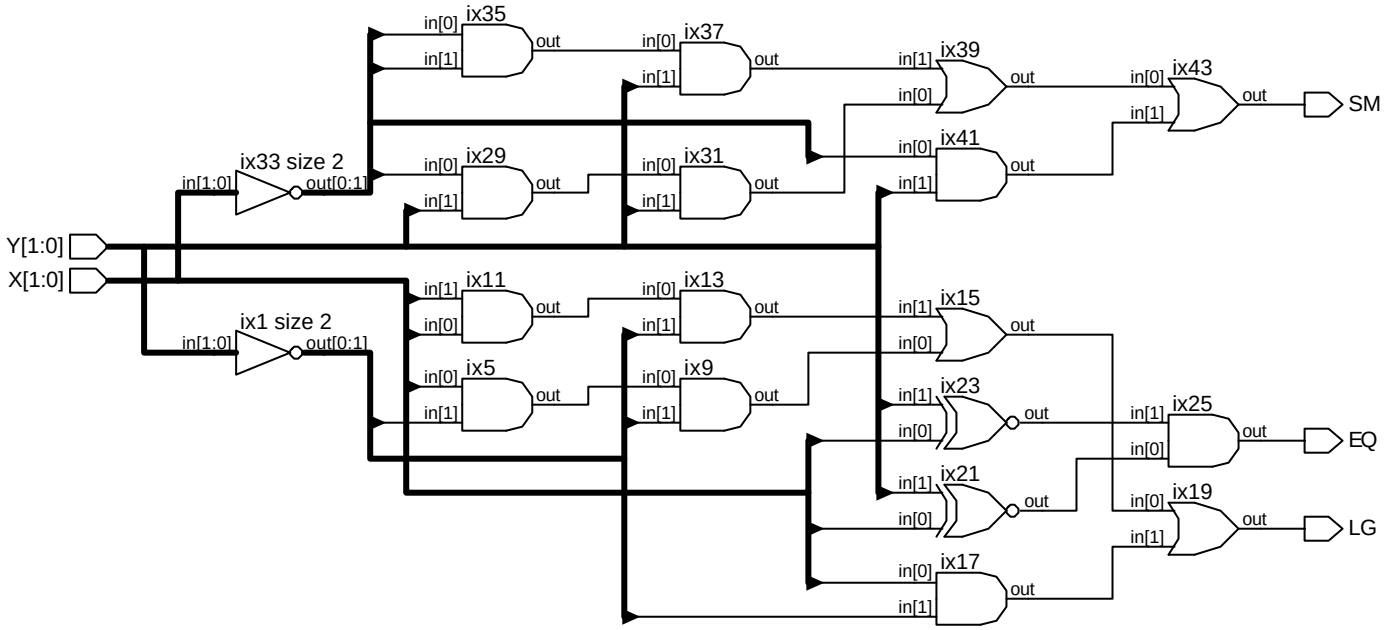

SEL
sel
ix19
A
in0
B
in1
out
OUT

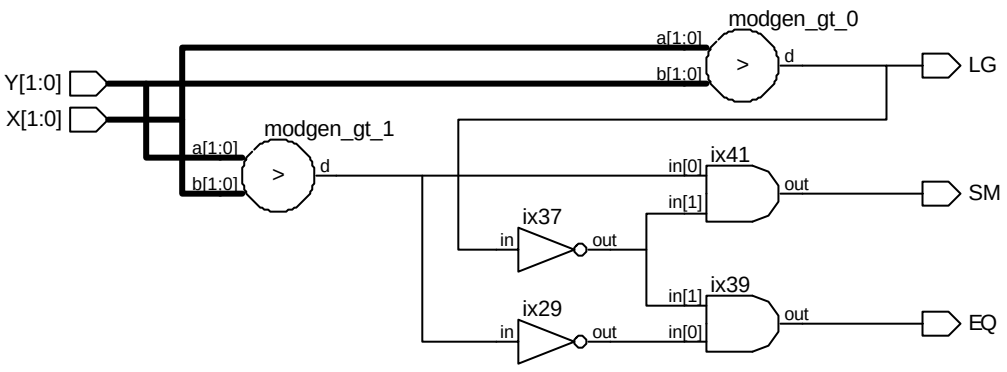

SEL[1:0]
A
B
C
D
sel
ix31
in0
in1
out
sel
ix33
in0
in1
out
sel
ix35
in0
in1
out
OUT

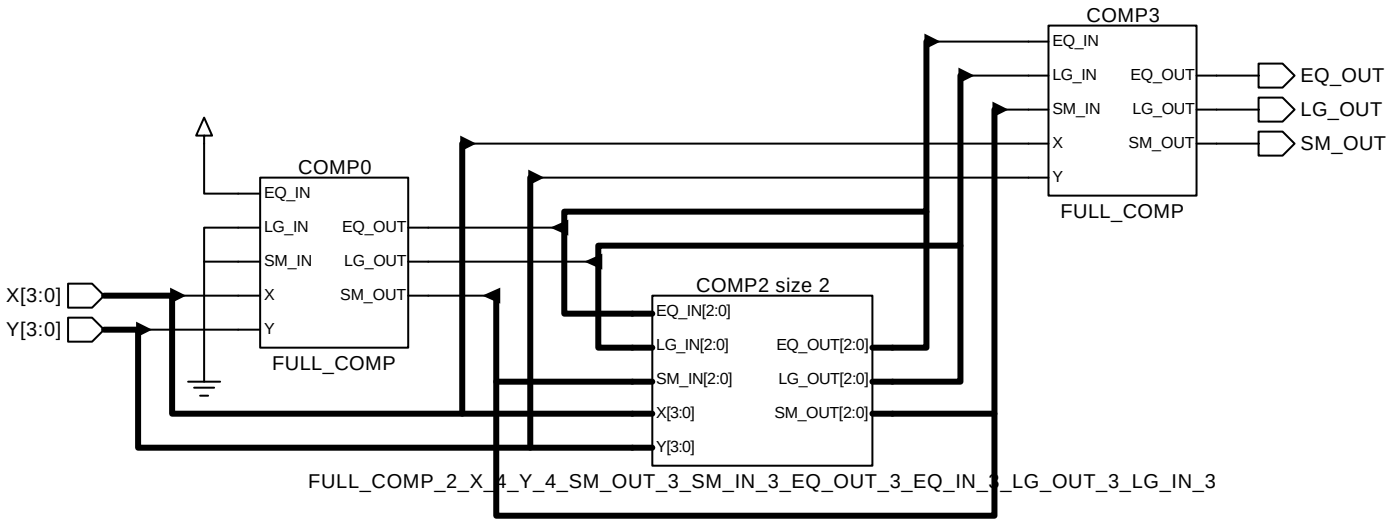

B
C
D
A
SEL[1:0]
modgen_mux_4
OUT

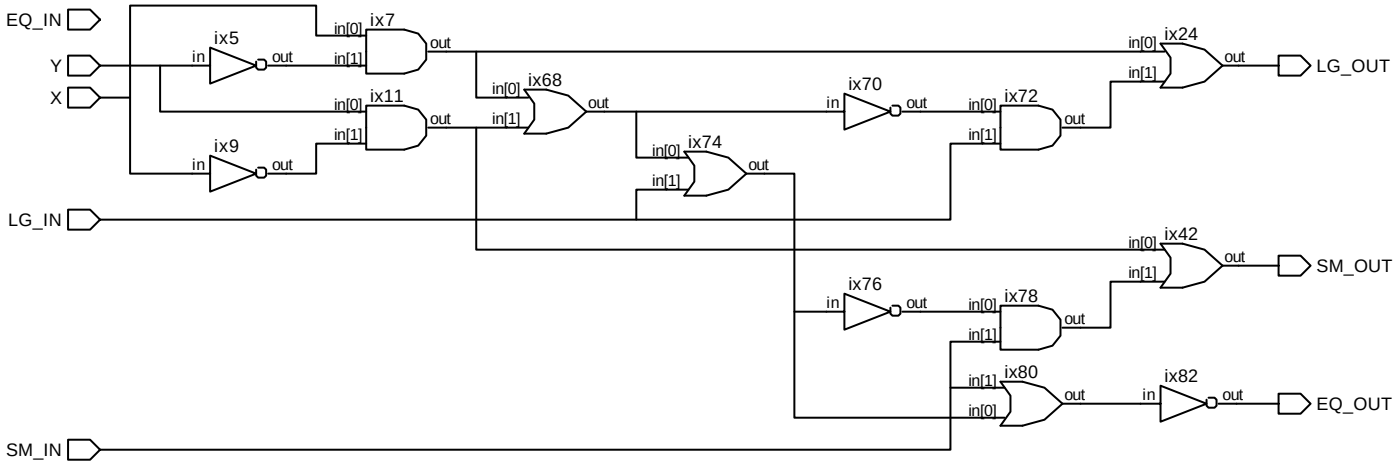

C[1:0]
D[1:0]
SEL[1:0]
A[1:0]
B[1:0]
sel
in0
in1
out
ix51
ix49
ix45
ix47
ix53
ix55
OUT[1:0]

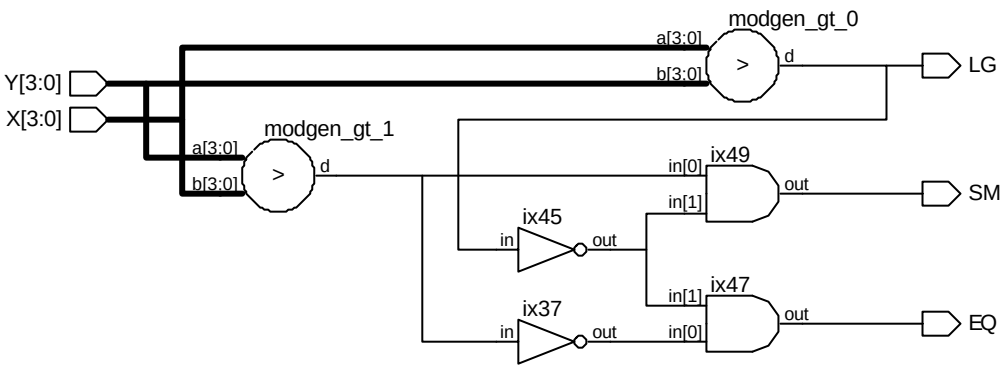

modgen_mux_5
SEL[1:0]
D[1:0]
B[1:0]
C[1:0]
A[1:0]
modgen_mux_4
OUT[1:0]

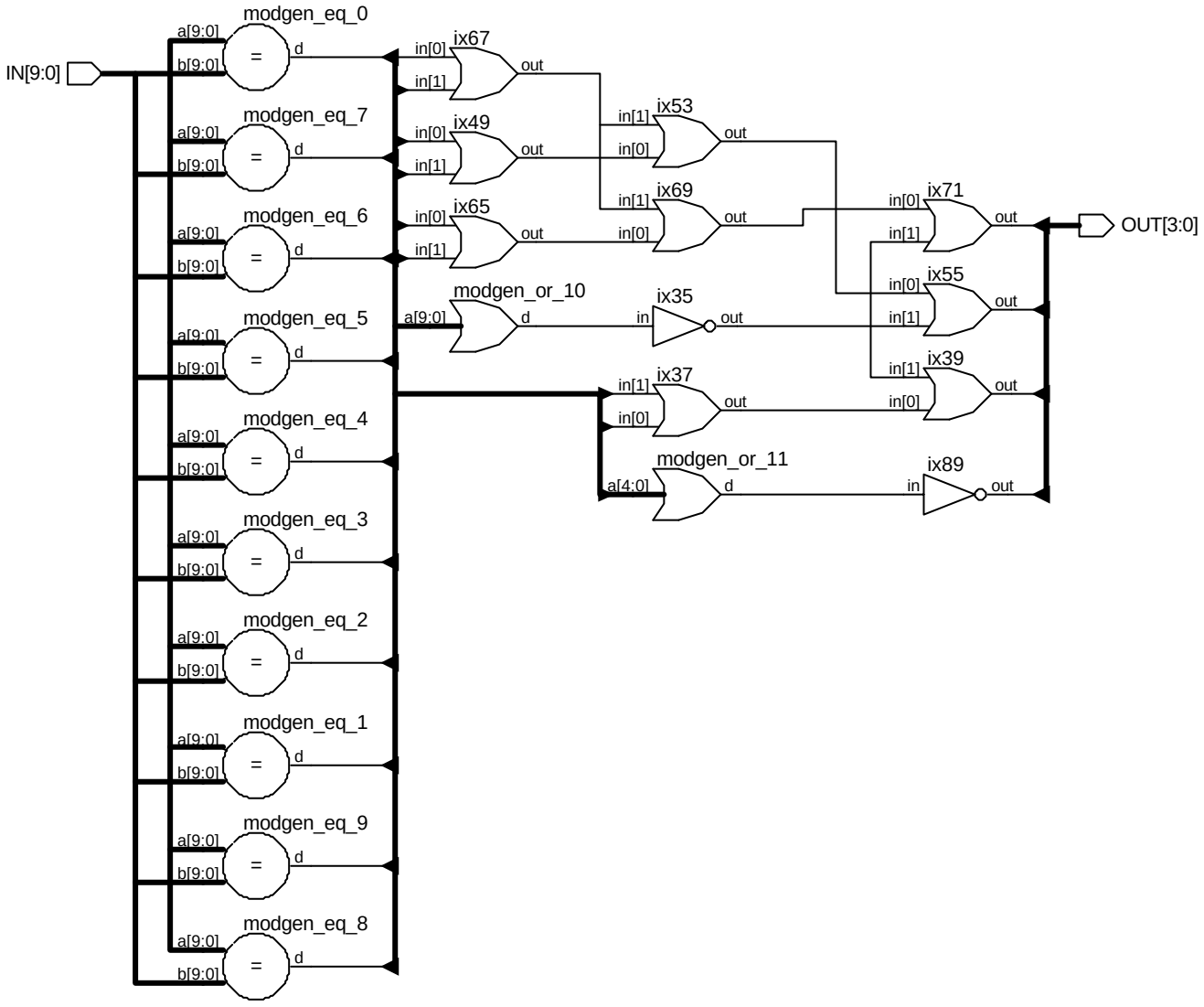

Y[1:0]
X[1:0]
ix33 size 2
in[1:0]
out[0:1]
ix1 size 2
in[1:0]
out[0:1]
ix35
ix37
ix39
ix43
ix29
ix31
ix41
ix11
ix13
ix15
ix5
ix9
ix23
ix25
ix21
ix19
ix17
in[0]
in[1]
out
SM
EQ
LG

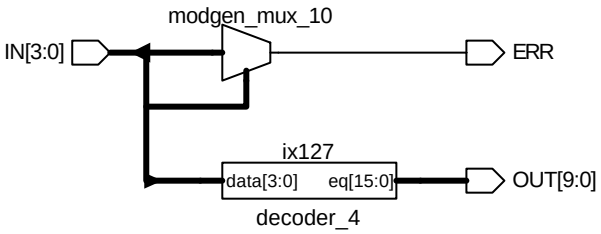

modgen_gt_0
a[1:0]
b[1:0]
>
d
LG
Y[1:0]
X[1:0]
modgen_gt_1
a[1:0]
b[1:0]
>
d
ix41
in[0]
in[1]
out
SM
ix37
in
out
ix39
in[1]
out
ix29
in
out
in[0]
EQ

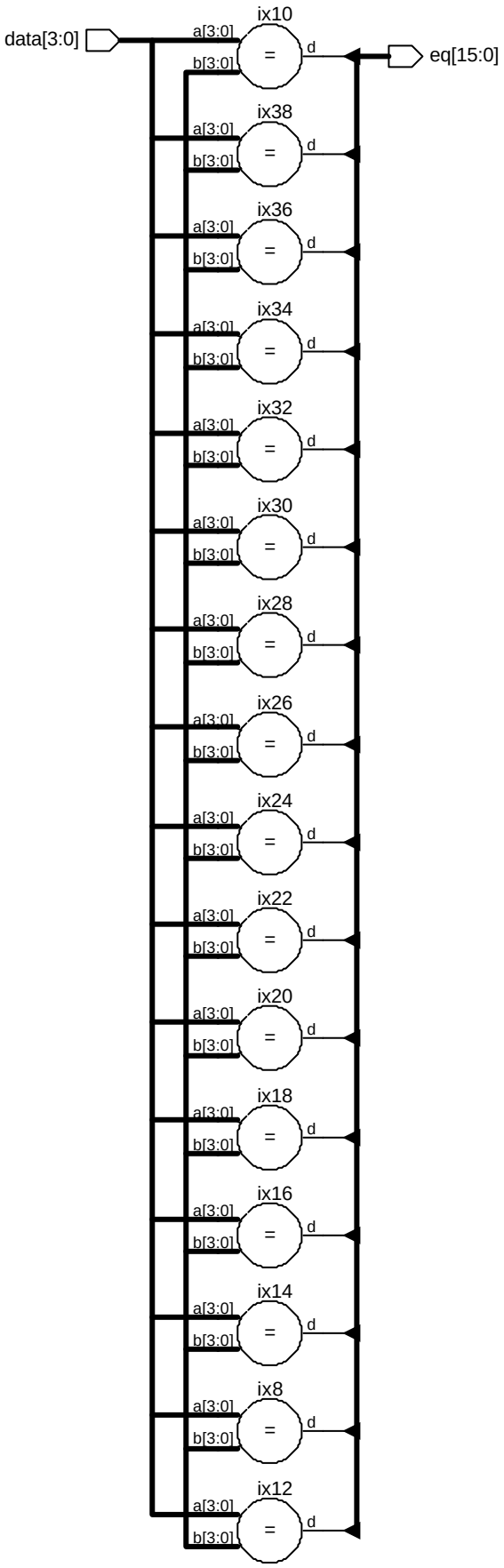

COMP3
EQ_IN
LG_IN
SM_IN
X
Y
EQ_OUT
LG_OUT
SM_OUT
FULL_COMP
EQ_OUT
LG_OUT
SM_OUT
COMP0
EQ_IN
LG_IN
SM_IN
X
Y
EQ_OUT
LG_OUT
SM_OUT
FULL_COMP
X[3:0]
Y[3:0]
COMP2 size 2
EQ_IN[2:0]
LG_IN[2:0]
SM_IN[2:0]
X[3:0]
Y[3:0]
EQ_OUT[2:0]
LG_OUT[2:0]
SM_OUT[2:0]
FULL_COMP_2_X_4_Y_4_SM_OUT_3_SM_IN_3_EQ_OUT_3_EQ_IN_3_LG_OUT_3_LG_IN_3

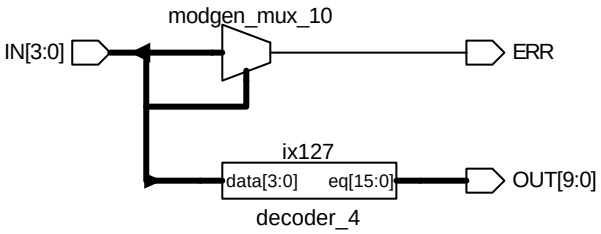

EQ_IN
Y
X
LG_IN
SM_IN
ix5
in
out
ix9
in
out
ix7
in[0]
in[1]
out
ix11
in[0]
in[1]
out
ix68
in[0]
in[1]
out
ix74
in[0]
in[1]
out
ix70
in
out
ix72
in[0]
in[1]
out
ix24
in[0]
in[1]
out
LG_OUT
ix76
in
out
ix78
in[0]
in[1]
out
ix42
in[0]
in[1]
out
SM_OUT
ix80
in[1]
in[0]
out
ix82
in
out
EQ_OUT

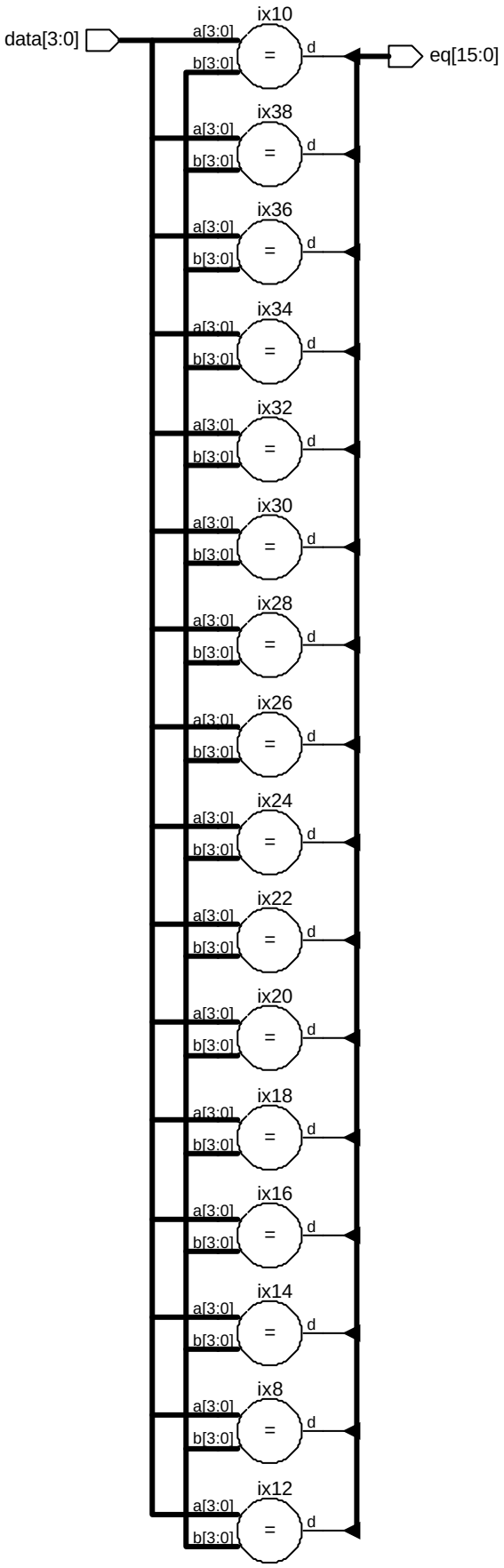

modgen_gt_0
a[3:0]
b[3:0]
>
d
LG
Y[3:0]
X[3:0]
modgen_gt_1
a[3:0]
b[3:0]
>
d
ix49
in[0]
in[1]
out
SM
ix45
in
out
ix47
in[1]
out
ix37
in
out
in[0]
EQ

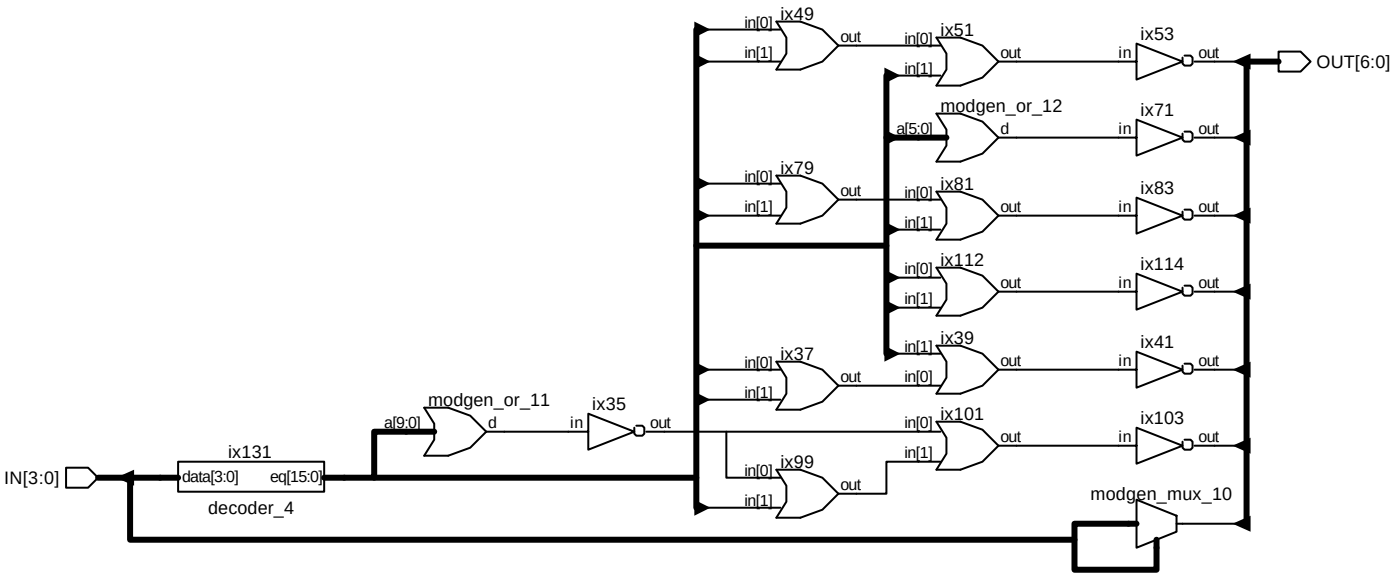

IN[9:0]
modgen_eq_0
modgen_eq_7
modgen_eq_6
modgen_eq_5
modgen_eq_4
modgen_eq_3
modgen_eq_2
modgen_eq_1
modgen_eq_9
modgen_eq_8
a[9:0]
b[9:0]
=
d
ix67
ix49
ix65
ix53
ix69
ix71
ix55
ix39
ix37
ix35
ix89
in[0]
in[1]
out
modgen_or_10
modgen_or_11
a[4:0]
in
OUT[3:0]

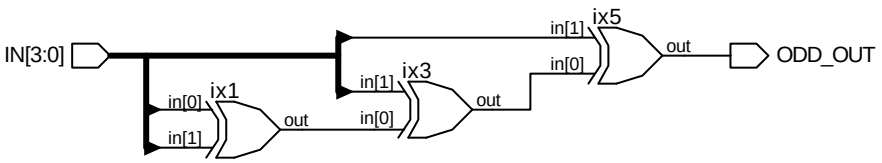

modgen_mux_10
IN[3:0]
ERR
ix127
data[3:0]
eq[15:0]
OUT[9:0]
decoder_4

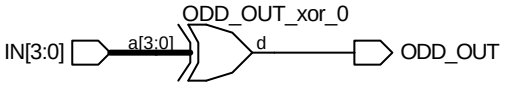

data[3:0]
eq[15:0]
ix10
ix38
ix36
ix34
ix32
ix30
ix28
ix26
ix24
ix22
ix20
ix18
ix16
ix14
ix8
ix12
a[3:0]
b[3:0]
=
d

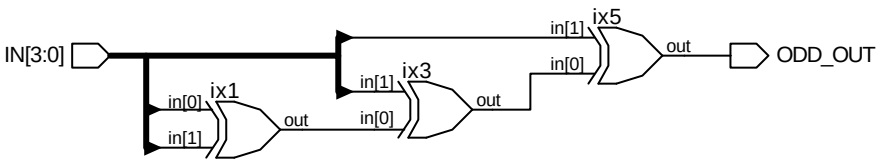

modgen_mux_10
IN[3:0]
ERR
ix127
data[3:0]
eq[15:0]
OUT[9:0]
decoder_4

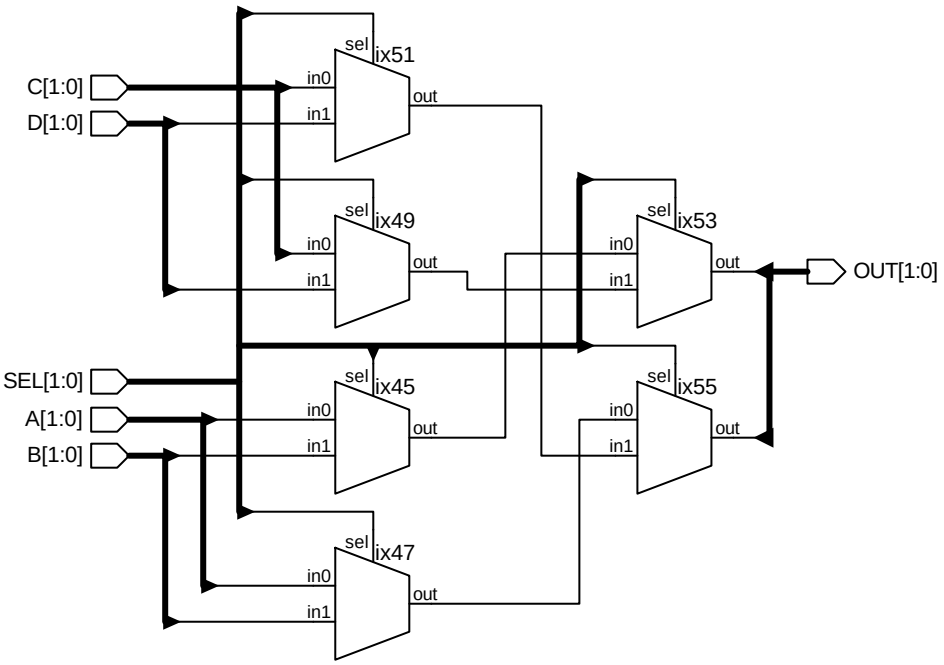

data[3:0]
eq[15:0]
ix10
ix38
ix36
ix34
ix32
ix30
ix28
ix26
ix24
ix22
ix20
ix18
ix16
ix14
ix8
ix12
a[3:0]
b[3:0]
=
d

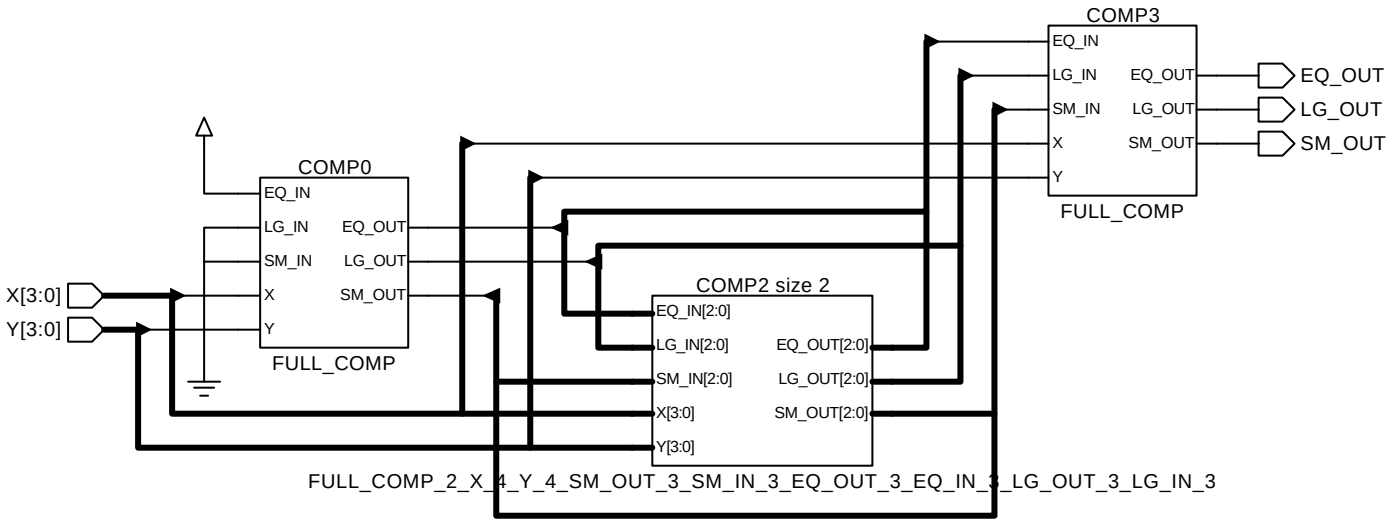

IN[3:0]
ix131
data[3:0]
eq[15:0]
decoder_4
modgen_or_11
a[9:0]
d
ix35
in
out
ix49
in[0]
in[1]
out
ix79
ix37
ix99
ix51
ix81
ix39
ix101
modgen_or_12
a[5:0]
ix112
ix53
ix71
ix83
ix114
ix41
ix103
modgen_mux_10
OUT[6:0]

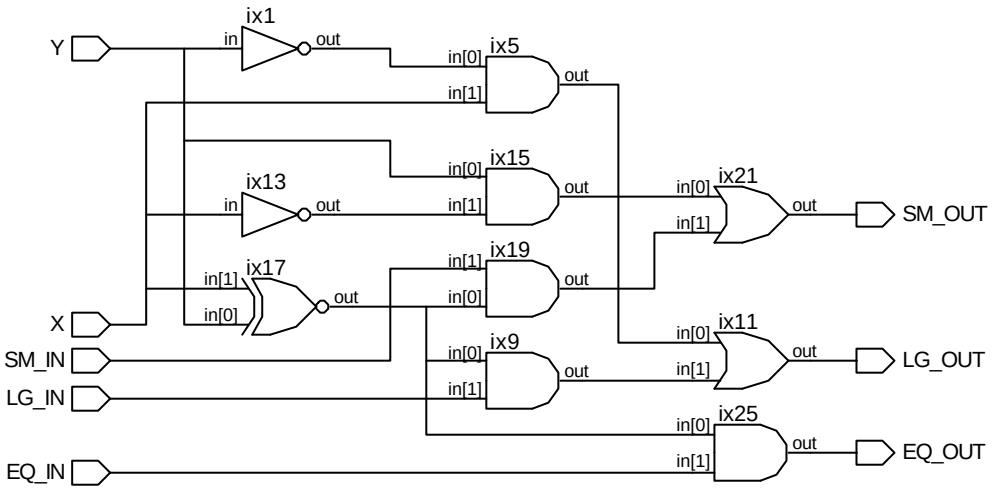

IN[3:0]
ix1
in[0]
in[1]
out
ix3
in[1]
in[0]
out
ix5
in[1]
in[0]
out
ODD_OUT

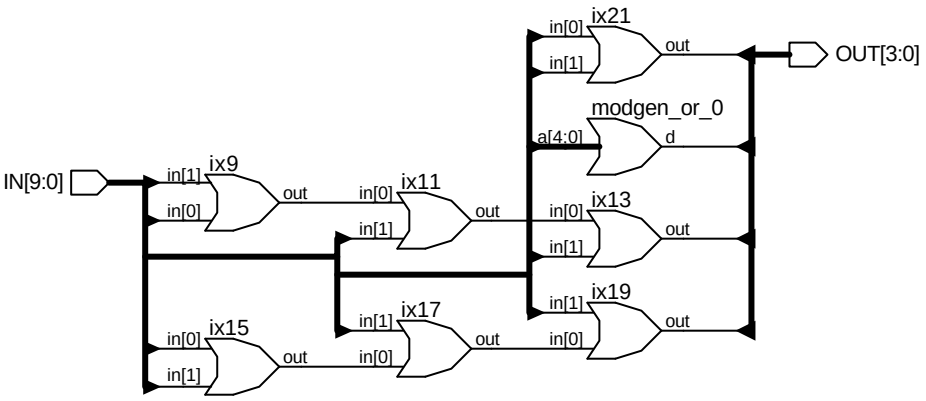

ODD_OUT_xor_0
IN[3:0]
a[3:0]
d
ODD_OUT

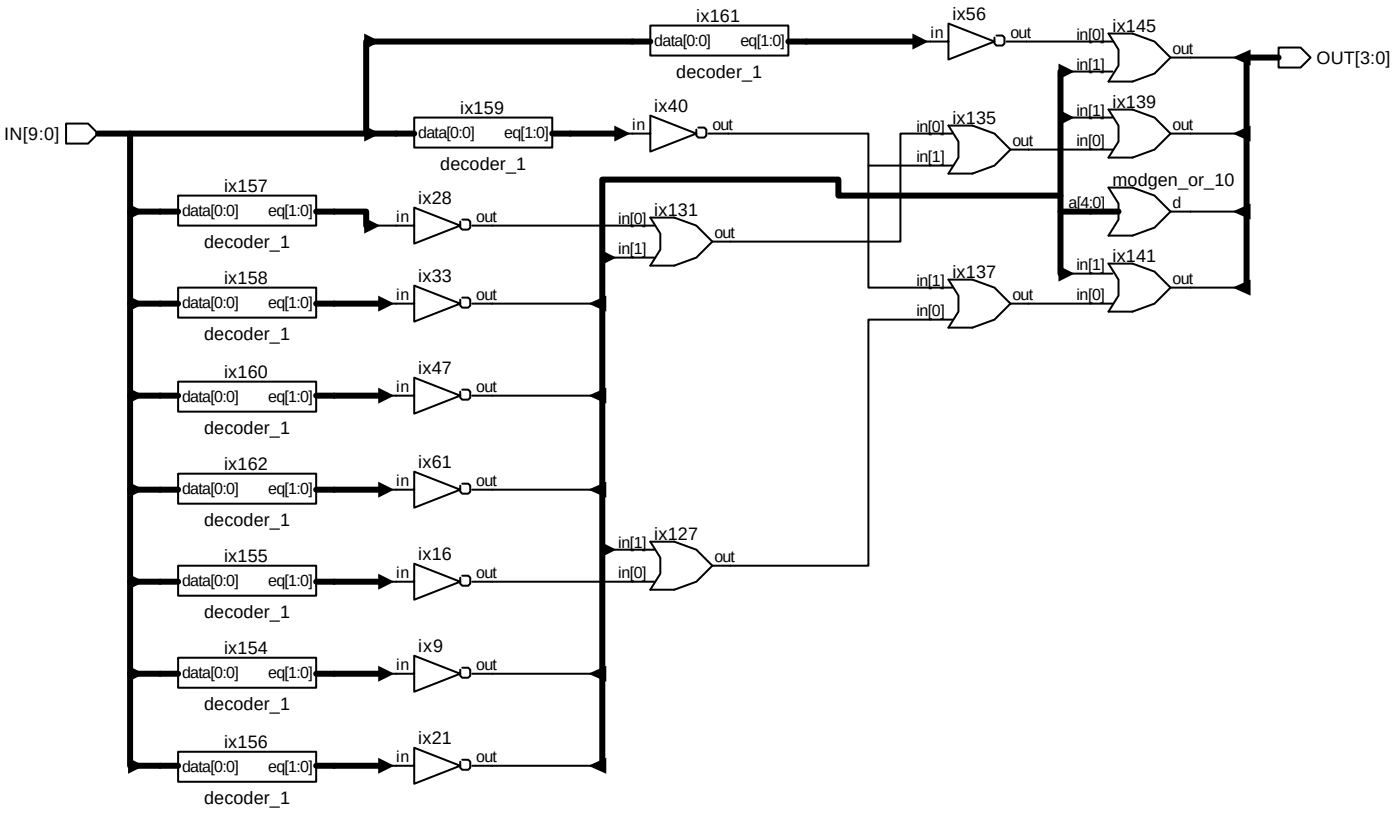

IN[3:0]
ix1
in[0]
in[1]
out
ix3
in[1]
in[0]
out
ix5
in[1]
in[0]
out
ODD_OUT

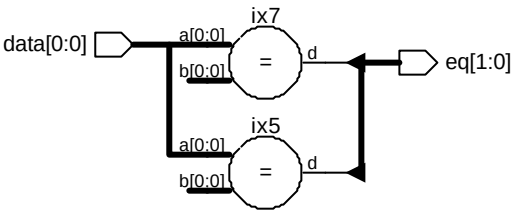

C[1:0]
D[1:0]
SEL[1:0]
A[1:0]
B[1:0]
sel
in0
in1
out
ix51
ix49
ix45
ix47
ix53
ix55
OUT[1:0]

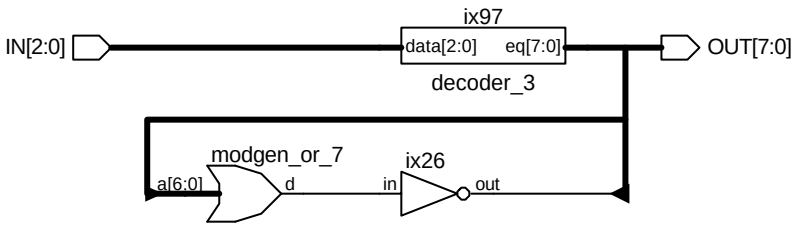

COMP3
EQ_IN
LG_IN
SM_IN
X
Y
EQ_OUT
LG_OUT
SM_OUT
FULL_COMP
EQ_OUT
LG_OUT
SM_OUT
COMP0
EQ_IN
LG_IN
SM_IN
X
Y
EQ_OUT
LG_OUT
SM_OUT
FULL_COMP
X[3:0]
Y[3:0]
COMP2 size 2
EQ_IN[2:0]
LG_IN[2:0]
SM_IN[2:0]
X[3:0]
Y[3:0]
EQ_OUT[2:0]
LG_OUT[2:0]
SM_OUT[2:0]
FULL_COMP_2_X_4_Y_4_SM_OUT_3_SM_IN_3_EQ_OUT_3_EQ_IN_3_LG_OUT_3_LG_IN_3

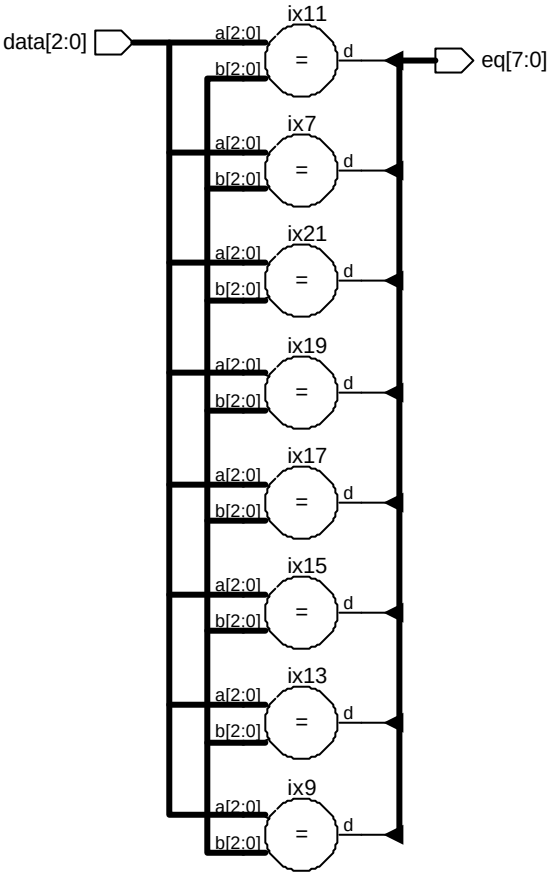

Y
X
SM_IN
LG_IN
EQ_IN
ix1
in
out
ix13
in
out
ix17
in[1]
in[0]
out
ix5
in[0]
in[1]
out
ix15
in[0]
in[1]
out
ix19
in[1]
in[0]
out
ix9
in[0]
in[1]
out
ix21
in[0]
in[1]
out
SM_OUT
ix11
in[0]
in[1]
out
LG_OUT
ix25
in[0]
in[1]
out
EQ_OUT


IN[9:0]
ix9
in[1]
in[0]
out
ix11
in[0]
in[1]
out
ix15
in[0]
in[1]
out
ix17
in[1]
in[0]
out
ix21
in[0]
in[1]
out
modgen_or_0
a[4:0]
d
ix13
in[0]
in[1]
out
ix19
in[1]
in[0]
out
OUT[3:0]


IN[9:0]
OUT[3:0]
ix161
data[0:0] eq[1:0]
decoder_1
ix56
in
out
ix145
in[0]
in[1]
out
ix159
data[0:0] eq[1:0]
decoder_1
ix40
in
out
ix135
in[0]
in[1]
out
ix139
in[1]
in[0]
out
modgen_or_10
a[4:0]
d
ix157
data[0:0] eq[1:0]
decoder_1
ix28
in
out
ix131
in[0]
in[1]
out
ix158
data[0:0] eq[1:0]
decoder_1
ix33
in
out
ix137
in[1]
in[0]
out
ix141
in[1]
in[0]
out
ix160
data[0:0] eq[1:0]
decoder_1
ix47
in
out
ix162
data[0:0] eq[1:0]
decoder_1
ix61
in
out
ix155
data[0:0] eq[1:0]
decoder_1
ix16
in
out
ix127
in[1]
in[0]
out
ix154
data[0:0] eq[1:0]
decoder_1
ix9
in
out
ix156
data[0:0] eq[1:0]
decoder_1
ix21
in
out


data[0:0]
ix7
a[0:0]
b[0:0]
=
d
eq[1:0]
ix5
a[0:0]
b[0:0]
=
d


ix97
IN[2:0]
data[2:0]
eq[7:0]
OUT[7:0]
decoder_3
modgen_or_7
ix26
a[6:0]
d
in
out


data[2:0]
ix11
a[2:0]
b[2:0]
=
d
eq[7:0]
ix7
ix21
ix19
ix17
ix15
ix13
ix9