

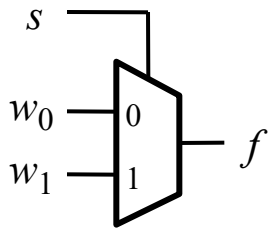
## 07 Combinational circuit elements

- Multiplexers (MUX's or selectors)
  - Crossbar and LUT examples
- Decoders
  - One-hot decoder with enable
  - Demultiplexers (DeMUX)
- Encoders/Priority Encoders
- Arithmetic comparators
- Verilog for combinational circuits

# Multiplexers:

A Multiplexer (mux for short) or a selector selects the output to follow one of many inputs:

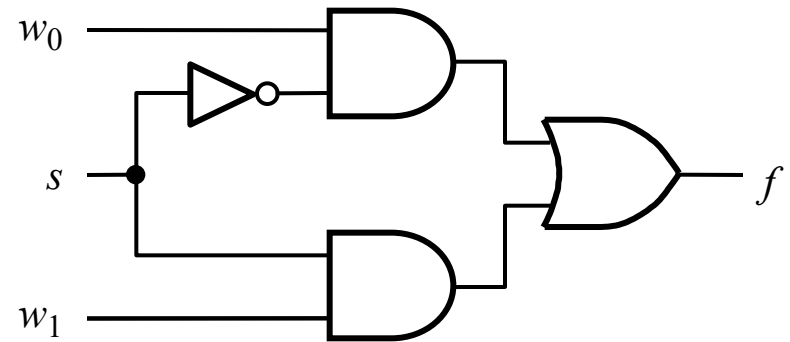
Example 1: a 2-to-1 mux:



Graphical symbol

$s$	$f$
0	$w_0$
1	$w_1$

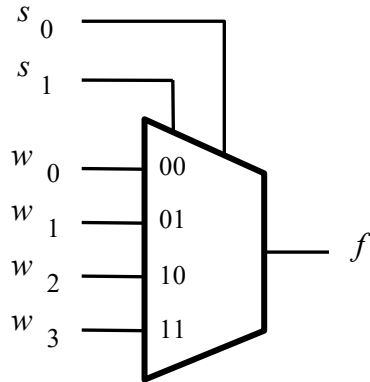
Truth table



Sum-of-products circuit

# Multiplexers (cont):

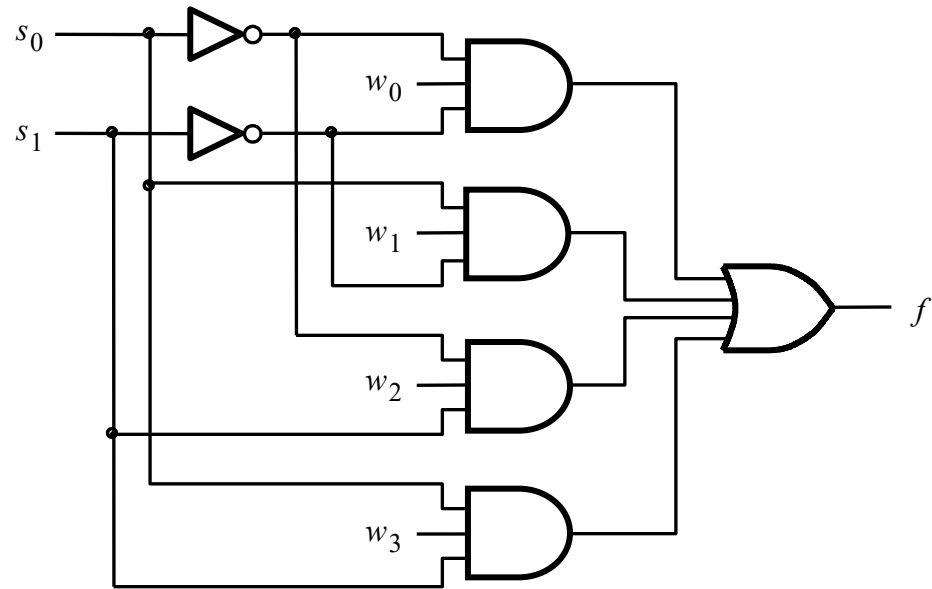
## Example 2: a 4-to-1 mux



Graphic symbol

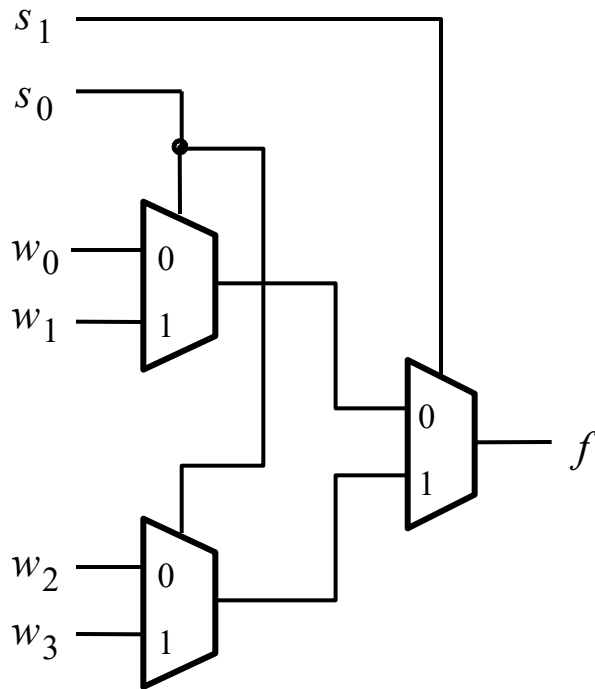
$s_1$	$s_0$	$f$
0	0	$w_0$
0	1	$w_1$
1	0	$w_2$
1	1	$w_3$

Truth table

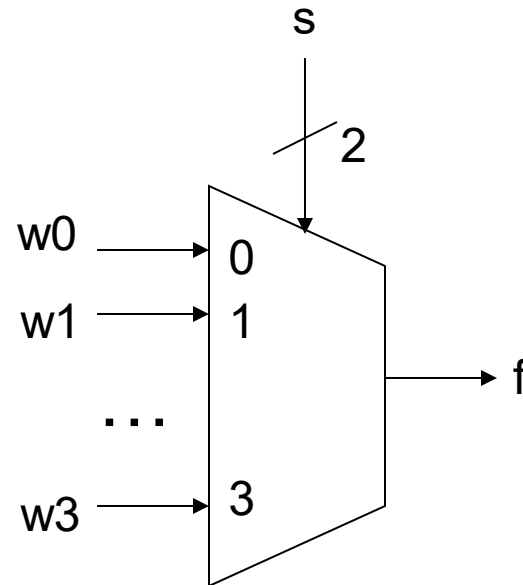


Circuit

# Building a 4-to-1 MUX using 2-to-1 MUX's:



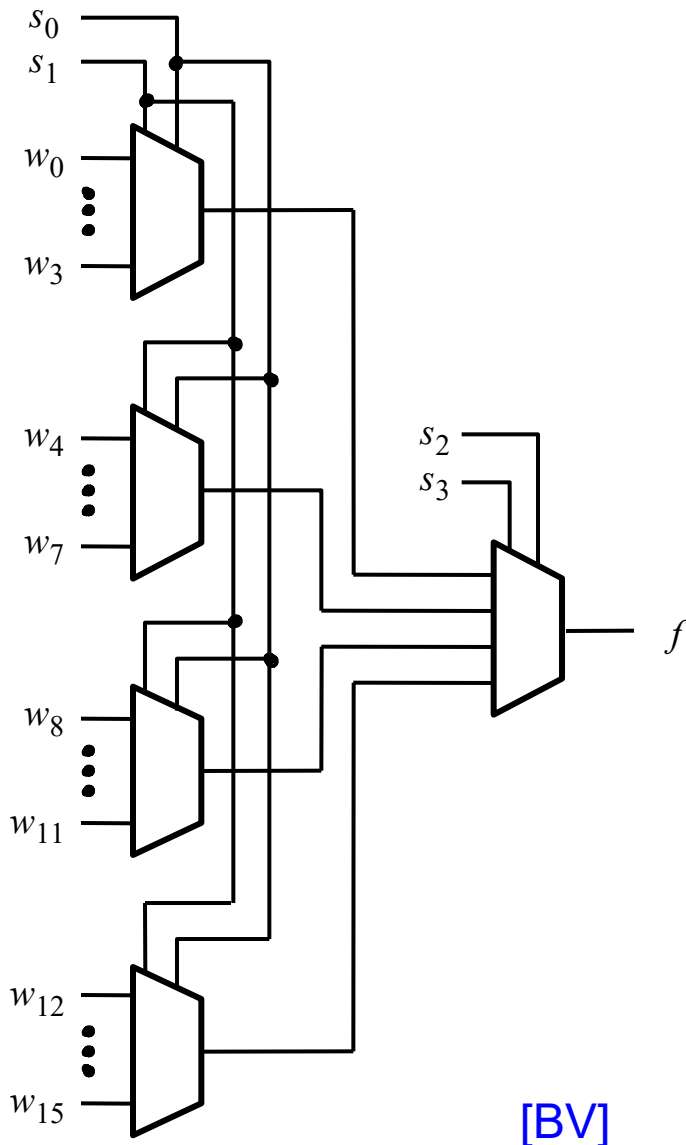
[BV]



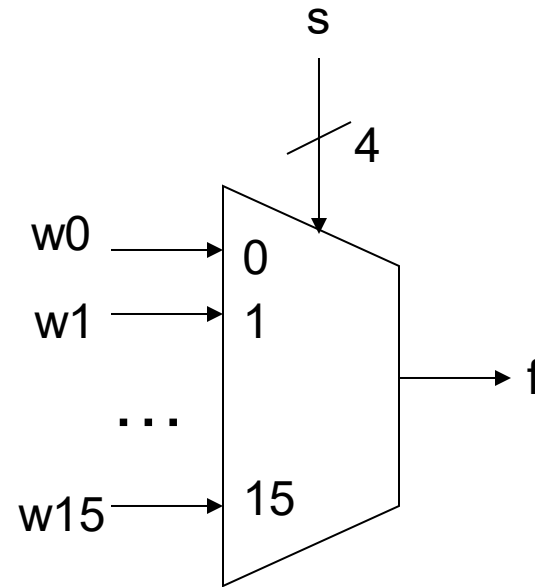
[K]

A select line ( $s$ ) which is 2-bit wide means  $s$  can have 4 different values  
→ selects from  $w_0 \dots w_3$  (4 inputs)

# Building a 16-to-1 MUX using 4-to-1 MUX's:



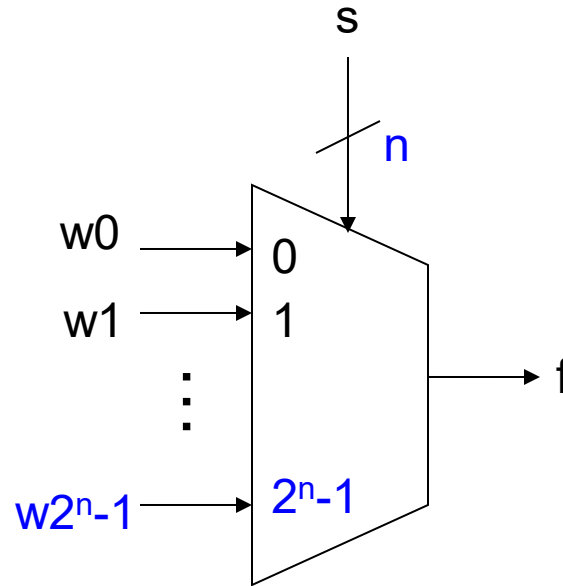
[BV]



[K]

A select line ( $s$ ) which is 4-bit wide means  $s$  can have 16 different values  
→ selects from  $w_0 \dots w_{15}$  (16 inputs)

## A mux with n-bit select lines:

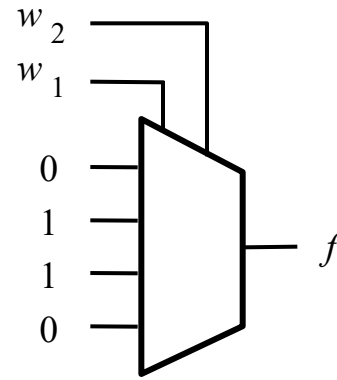


A select line (s) which is n-bit wide means s can have  $2^n$  different values  
→ selects from  $w_0 \dots w_{2^n-1}$  ( $2^n$  inputs)

Ex:  $2^6 = 64$  or  $6 = \log_2(64)$ , so  
a 64-to-1 mux has 6-bit wide select line

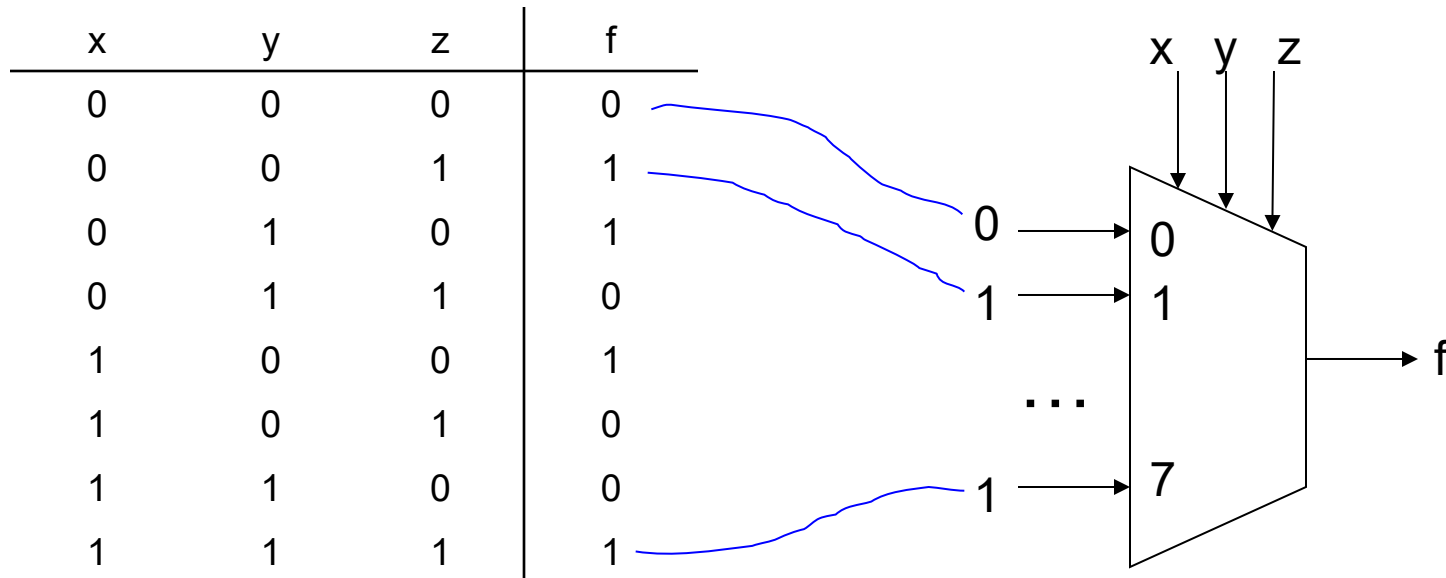
# Implementing truth table using MUX:

$w_1$	$w_2$	$f$
0	0	0
0	1	1
1	0	1
1	1	0



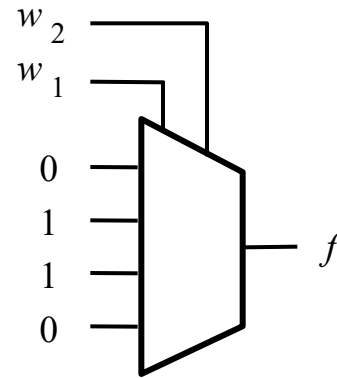
Implementation using a 4-to-1 multiplexer

can be extended to a function of more than 2 variables:



# Look-up table (LUT):

$w_1$	$w_2$	$f$
0	0	0
0	1	1
1	0	1
1	1	0



- The above can implement ANY function of 2 variables.  
(no need to do any function minimization at all)
- Extension to  $> 2$  variables possible.
- Efficient implementation of this “method” is called LUT
- Look-up table can be used to implement any function:



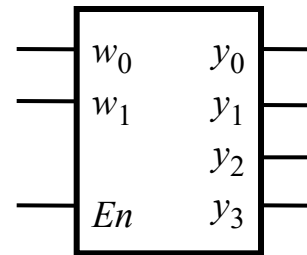


# Decoders:

## One-hot decoder with enable:

<i>En</i>	<i>w</i> <sub>1</sub>	<i>w</i> <sub>0</sub>	<i>y</i> <sub>0</sub>	<i>y</i> <sub>1</sub>	<i>y</i> <sub>2</sub>	<i>y</i> <sub>3</sub>
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

Truth table



Graphical symbol

2 inputs can have  $2^2 = 4$  possible input combinations

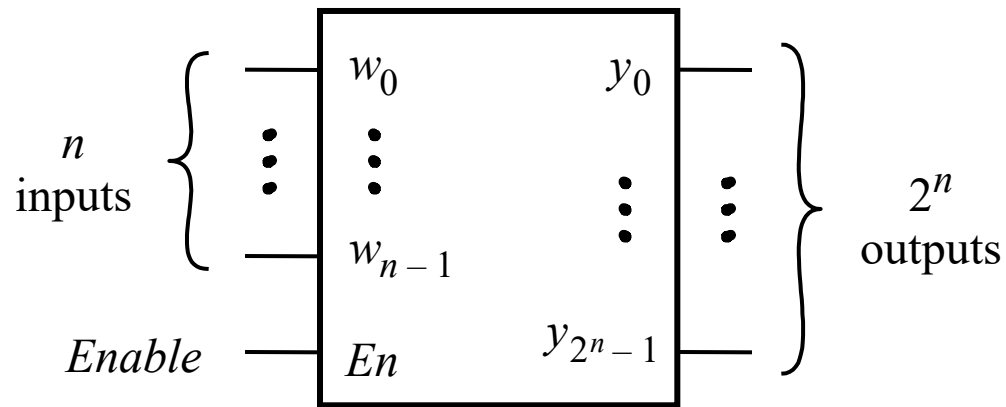
if (not enable) then all outputs are zero,

else (this means enable is 1)

assert one output indexed by input

# Decoders:

## One-hot decoder with enable:



- $n$  inputs can have  $2^n$  possible input combinations
- if (enable is 0)

All output lines are zero

else (this means enable is 0)

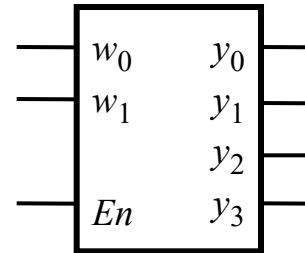
Assert only one  $y$  output indexed by the input.

 This is why it's called "one-hot." The rest of output is 0.

# Encoders:


Decoder: code to one-hot output:

$En$	$w_1$	$w_0$	$y_0$	$y_1$	$y_2$	$y_3$
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

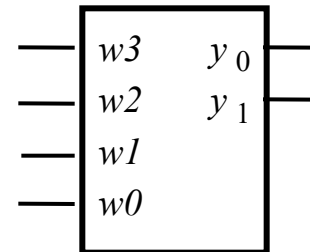


Encoder: one-hot input to code:

$w_3$	$w_2$	$w_1$	$w_0$	$y_1$	$y_0$
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



incomplete

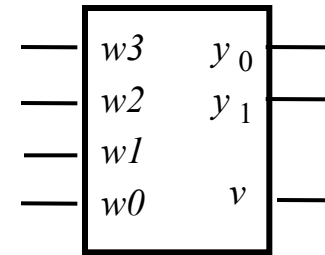


Graphical symbol

# Priority encoder:

$w_3$	$w_2$	$w_1$	$w_0$	$y_1$	$y_0$	$v$
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

$v$  indicates output  $y$  is valid



Graphical symbol

[BV]

$w_3$  has highest priority,  $w_0$  has lowest priority.

Real life example: We have only one server to serve 4 important people.

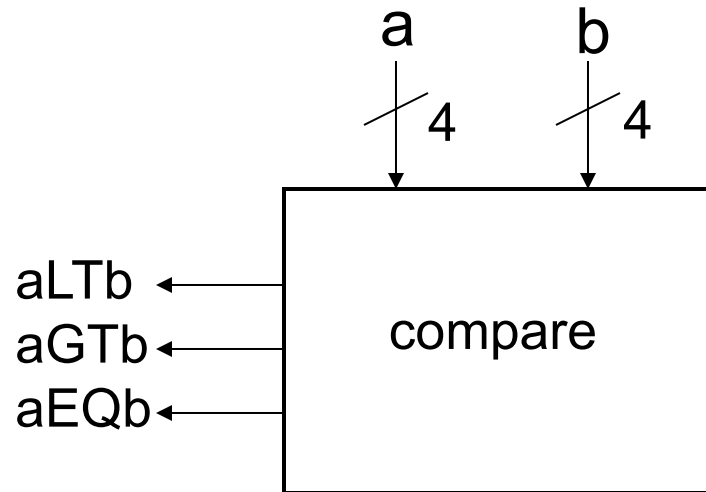
$w_3$  = most important, ...,  $w_0$  = least important

$v == 1 \rightarrow$  we need to serve someone,  
 $y$  is the code of the most important person to serve.

# Arithmetic comparators:

Compare the size of two numbers:

Example: 4-bit comparators



usually compares  
unsigned numbers.

Only 3 outcomes are possible:

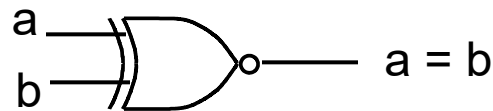
- |                   |                                |
|-------------------|--------------------------------|
| 1. $a < b$ (aLTb) | Ex: $a = 4'b1011, b = 4'b1101$ |
| 2. $a > b$ (aGTb) | Ex: $a = 4'b1000, b = 4'b0101$ |
| 3. $a = b$ (aEQb) | Ex: $a = 4'b0110, b = 4'b0110$ |

# Arithmetic comparators:

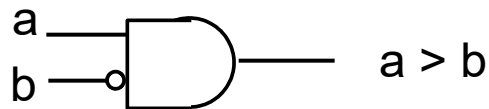
Consider 2 one-bit inputs:

a	b	a>b	a<b	a=b
0	0			1
0	1		1	
1	0	1		
1	1			1

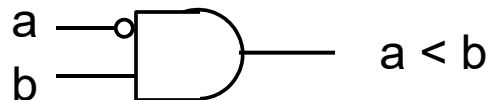
a = b is this:



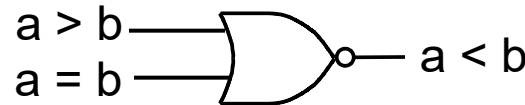
a > b is this:



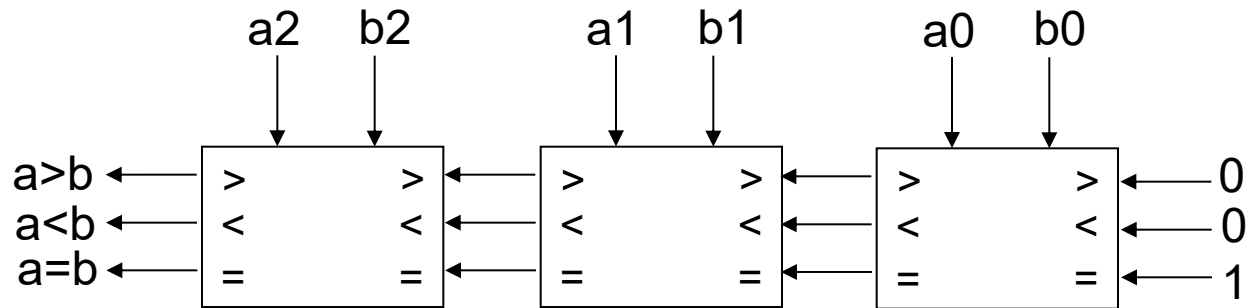
a < b is



or  $a < b = (aGTb + aEQb)'$



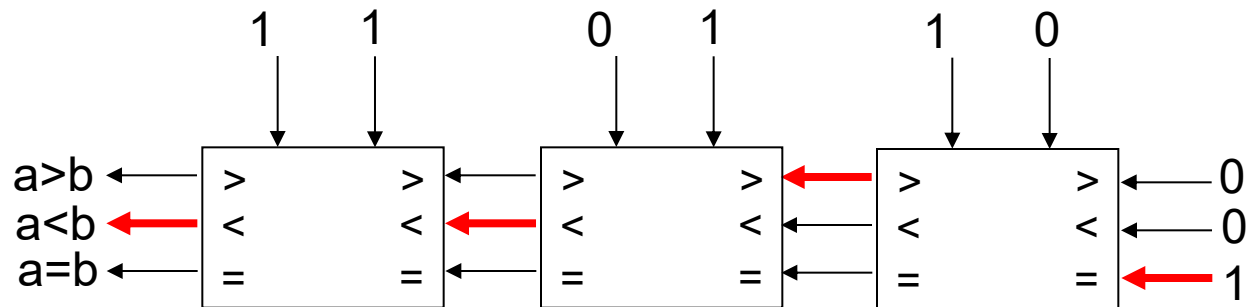
# The cascading comparator:



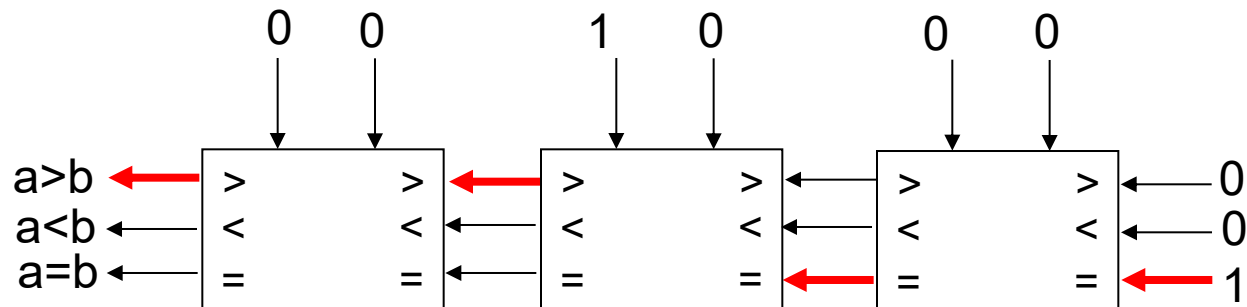
a and b		Cascade inputs			Outputs		
a,b	note	>	<	=	>	<	=
10	$a > b$	x	x	x	1	0	0
01	$a < b$	x	x	x	0	1	0
00 11	$a = b$	$>_{in}$	$<_{in}$	$=_{in}$	$>_{in}$	$<_{in}$	$=_{in}$

# The cascading comparator examples:

$a = 101, b = 110 \ (a < b)$



$a = 010, b = 000 \ (a > b)$

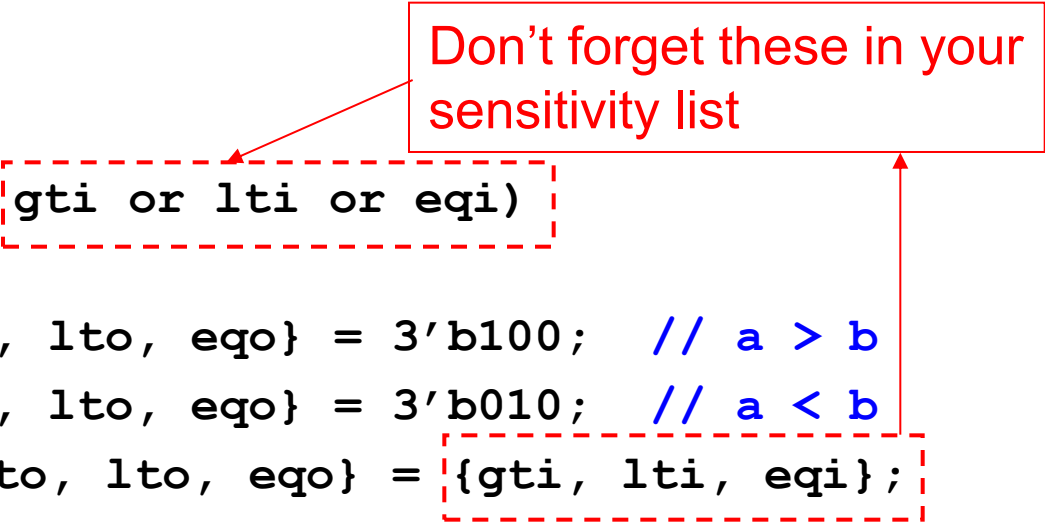




# Verilog for combinational logic:

## 1-bit cascading comparator:

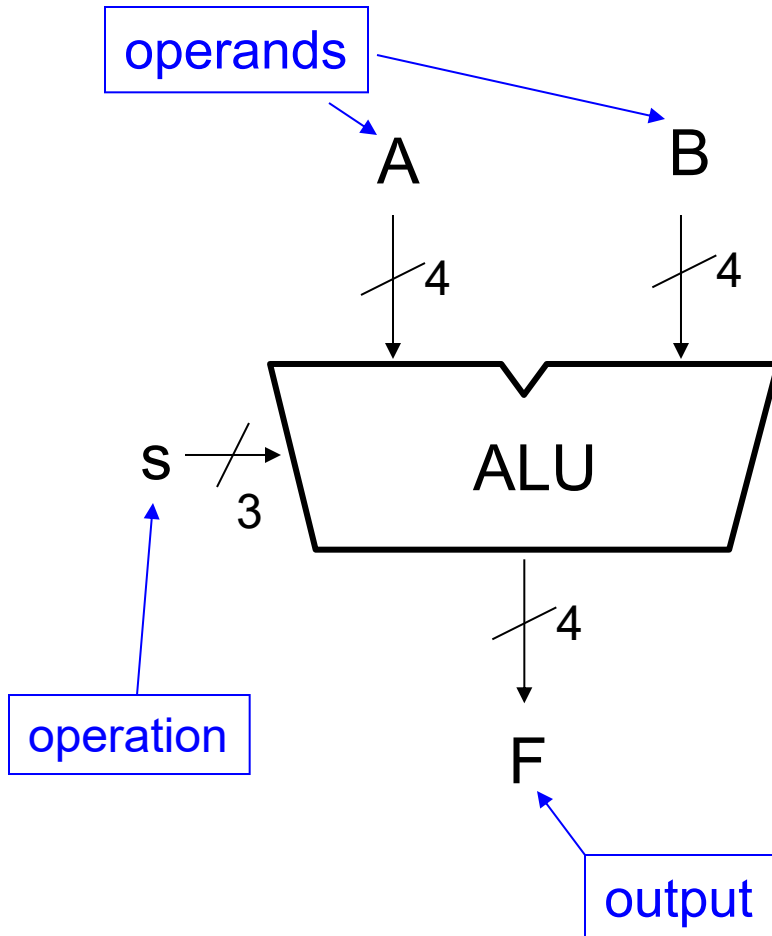
```
module comp (a, b, gti, lti, eqi, gto, lto, eqo);  
    input a, b, gti, lti, eqi;  
    output gto, lto, eqo;  
  
    reg gto, lto, eqo;  
  
    always @ (a or b or gti or lti or eqi)  
        case({a,b})  
            2'b10: {gto, lto, eqo} = 3'b100;    // a > b  
            2'b01: {gto, lto, eqo} = 3'b010;    // a < b  
            default: {gto, lto, eqo} = {gti, lti, eqi};  
        endcase  
  
endmodule
```



# Verilog for combinational logic:

## ALU: Arithmetic and logic unit

[BV]



Operation	Inputs			Outputs F
	$s_2$	$s_1$	$s_0$	
Clear	0	0	0	0 0 0 0
$B - A$	0	0	1	$B - A$
$A - B$	0	1	0	$A - B$
ADD	0	1	1	$A + B$
XOR	1	0	0	$A \text{ XOR } B$
OR	1	0	1	$A \text{ OR } B$
AND	1	1	0	$A \text{ AND } B$
Preset	1	1	1	1 1 1 1

The functionality of the 74381 ALU.

# ALU:

Operation	Inputs $s_2 \ s_1 \ s_0$	Outputs F
Clear	0 0 0	0 0 0 0
B−A	0 0 1	$B - A$
A−B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	$A \text{ XOR } B$
OR	1 0 1	$A \text{ OR } B$
AND	1 1 0	$A \text{ AND } B$
Preset	1 1 1	1 1 1 1

The functionality of the 74381 ALU.

[BV]

```
// 74381 ALU
module alu(s, A, B, F);
    input [2:0] s;
    input [3:0] A, B;
    output [3:0] F;
    reg [3:0] F;

    always @(s or A or B)
        case (s)
            0: F = 4'b0000;
            1: F = B - A;
            2: F = A - B;
            3: F = A + B;
            4: F = A ^ B;
            5: F = A | B;
            6: F = A & B;
            7: F = 4'b1111;
        endcase
endmodule
```

## A 4-to-1 MUX:

```
module mux4to1 (w, s, f);  
    input [3:0] w;  
    input [1:0] s;  
    output f;  
    reg f;  
  
    always @(w or s)  
        case (s)  
            0: f = w[0];  
            1: f = w[1];  
            2: f = w[2];  
            3: f = w[3];  
        endcase  
  
endmodule [BV/K]
```

## Some design notes

1. When creating a bus, always run the bit index from MSB to LSB (from high number to low number)
2. Always use lowercase letters for signal names.

# A 2-to-4 decoder:

```
module dec2to4 (w, y, en);  
    input [1:0] w;  
    input en;  
    output [3:0] y;  
    reg [3:0] y;  
  
    always @ (w or en)  
    begin  
        if (en == 0)  
            y = 4'b0000;  
        else  
            case (w)  
                0: y = 4'b0001;  
                1: y = 4'b0010;  
                2: y = 4'b0100;  
                3: y = 4'b1000;  
            endcase  
        end  
    end  
  
endmodule
```

[BV/K]

## Where to use uppercase?

There are used in defining  
*constants* in *text macros* or  
*constants* in *parameters*

Just like C programming  
practices.

# A priority encoder:

```
module priority (w, y, v);  
    input [3:0] w;  
    output [1:0] y; output v;  
    reg [1:0] y; reg v;  
  
    always @(w)  
    begin  
        v = 1;  
        if (w[3])  
            y = 3;  
        else if (w[2])  
            y = 2;  
        else if (w[1])  
            y = 1;  
        else if (w[0])  
            y = 0;  
        else begin  
            v = 0;  
            y = 2'bx;  
        end  
    end  
end  
endmodule
```

# A 4-bit comparator:

```
module compare
(a, b, a_eq_b, a_gt_b, a_lt_b);

    input [3:0] a, b;
    output a_eq_b, a_gt_b, a_lt_b;
    reg a_eq_b, a_gt_b, a_lt_b;

    always @(a or b)
    begin
        a_eq_b = 0;
        a_gt_b = 0;
        a_lt_b = 0;
        if (a == b)
            a_eq_b = 1;
        else if (a > b)
            a_gt_b = 1;
        else
            a_lt_b = 1;
        end
    endmodule
```