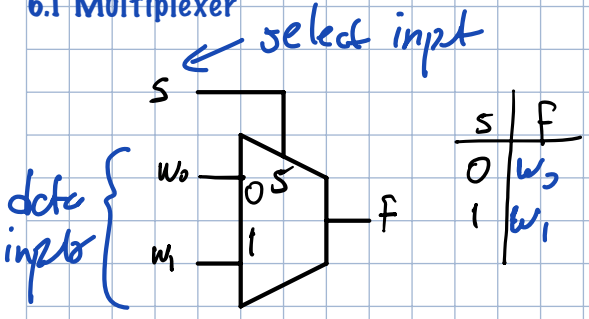
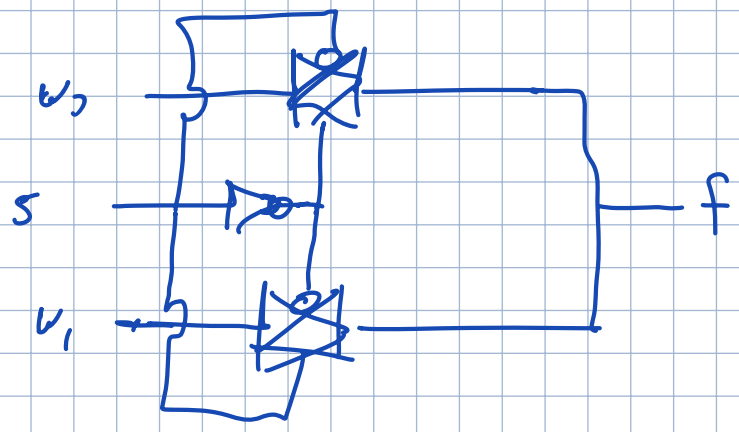
6.1 Multiplexer



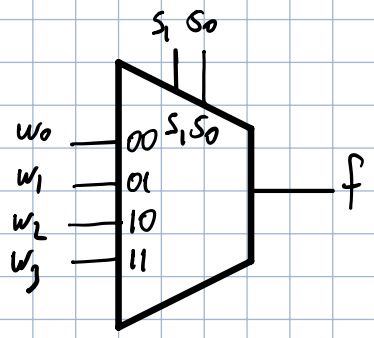
s	f
0	w_0
1	w_1

Cheapest way to construct 2-to-1 MUX?



requires 6 transistors

4 to 1 Multiplexer



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

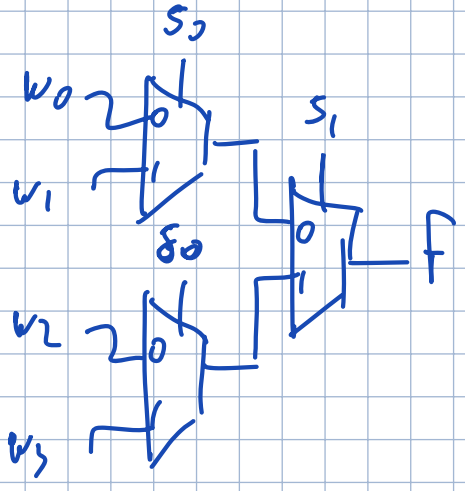
SOP of $f = \bar{s}_1 \bar{s}_0 w_0 + \bar{s}_1 s_0 w_1 + s_1 \bar{s}_0 w_2 + s_1 s_0 w_3$

AND-OR implementation: 4 3-input AND, 1 4-input OR, 2 inverters

8 + 10 + 4 = 46 transistors

4-to-1 MUX using AND, OR NOT gates: show as exercise

4-to-1 MUX " 2-to-1 MUXES

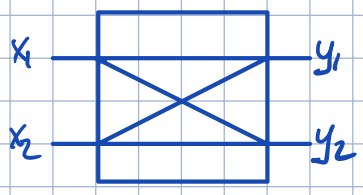


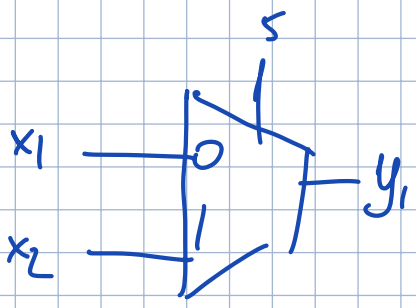
6x3=18 transistors

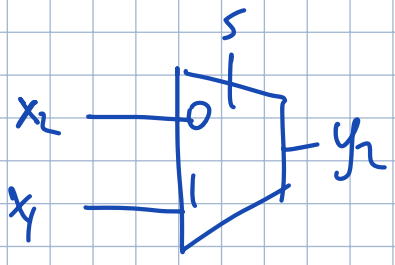
can be reduced to 16 if 1 inverter is used to get $\overline{s_0}$.

2x2 Crossbar Switch

s	y_1	y_2
0	x_1	x_2
1	x_2	x_1





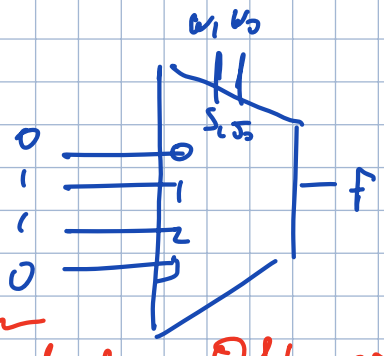


Implementation of logic functions using multiplexers

$f = w_1 \oplus w_2$

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

4-to-1 MUX

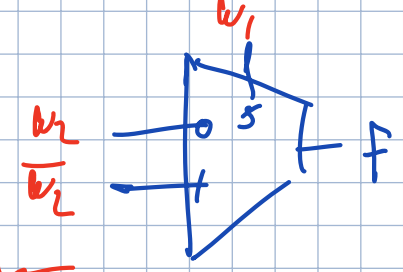


logic levels 0 & 1 are available

simpler implementation?

w_1	w_2	f	
0	0	0	w_2
0	1	1	
1	0	1	$\overline{w_2}$
1	1	0	

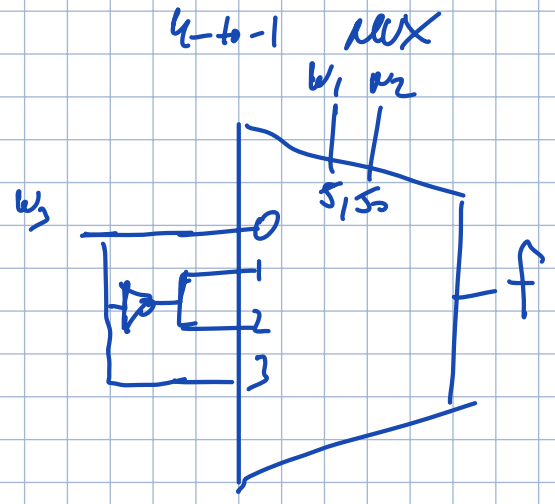
2-to-1 MUX



complements of input variables are available

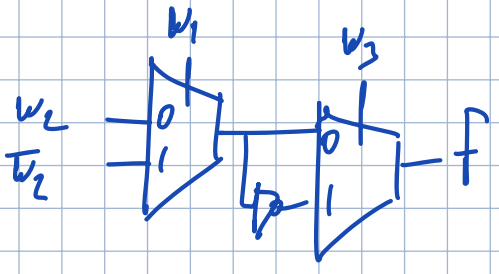
$f = w_1 \oplus w_2 \oplus w_3$ odd function

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\overline{w_3}$
0	1	1	0	
1	0	0	1	$\overline{w_3}$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	



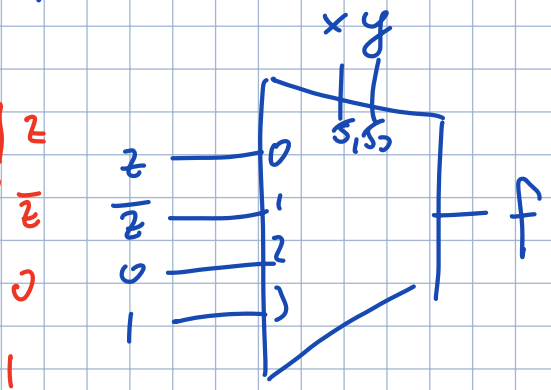
$$f = \underbrace{(w_1 \oplus w_2)}_{a} \oplus w_3 = \underbrace{0 \oplus w_3}_{\text{2-to-1 MUX}}$$

2-to-1 MUX



Exercise: $f(x,y,z) = \sum m(1,2,6,7)$

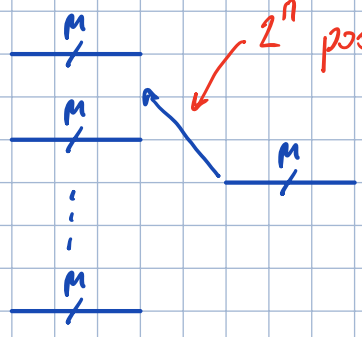
x	y	z	f	
0	0	0	0	z
0	0	1	1	
0	1	0	1	$\overline{z}$
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	

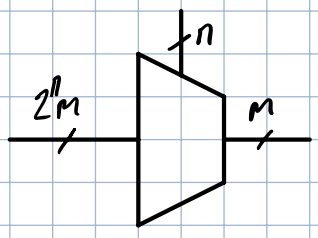


* In general, any n-input function can be implemented by 2^{n-1} to 1 MUX.

n-1 inputs go to select inputs
1 input goes to data inputs

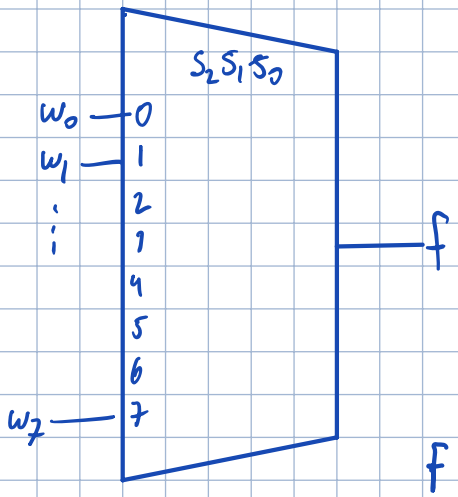
General form





2^n to 1 = 3 select bits

Design of 8 to 1 multiplexer in VHDL



S_2	S_1	S_0	f
0	0	0	w_0
0	0	1	w_1
0	1	0	w_2
0	1	1	w_3
1	0	0	w_4
1	0	1	w_5
1	1	0	w_6
1	1	1	w_7

$$f = \bar{S}_2\bar{S}_1\bar{S}_0 w_0 + \bar{S}_2\bar{S}_1 S_0 w_1 + \bar{S}_2 S_1 \bar{S}_0 w_2 + \cdots + S_2 S_1 S_0 w_7$$

```
entity mux8to1 is
    port(w: in std_logic_vector(7 downto 0);
         s: in std_logic_vector(2 downto 0);
         f: out std_logic);
end mux8to1;
```

```
architecture mux8to1_arch of mux8to1 is
begin
    with s select
        f <= w(0) when "000",
             w(1) when "001",
             w(2) when "010",
             w(3) when "011",
             w(4) when "100",
             w(5) when "101",
             w(6) when "110",
             w(7) when others;
end mux8to1_arch;
```

↳ all other values including 111.

S_2	S_1	S_0	f
0	0	0	w_0
0	0	1	w_1
0	1	0	w_2
0	1	1	w_3
1	0	0	w_4
1	0	1	w_5
1	1	0	w_6
1	1	1	w_7

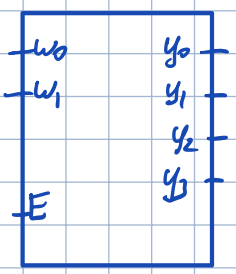
Decoders

2-to-4 decoder

enable input

E	w_1	w_0	y_3	y_2	y_1	y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

} one-hot



w_1	w_0	y_3	y_2	y_1	y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

(decoder without enable)

$y_i = m_i$ minterm i

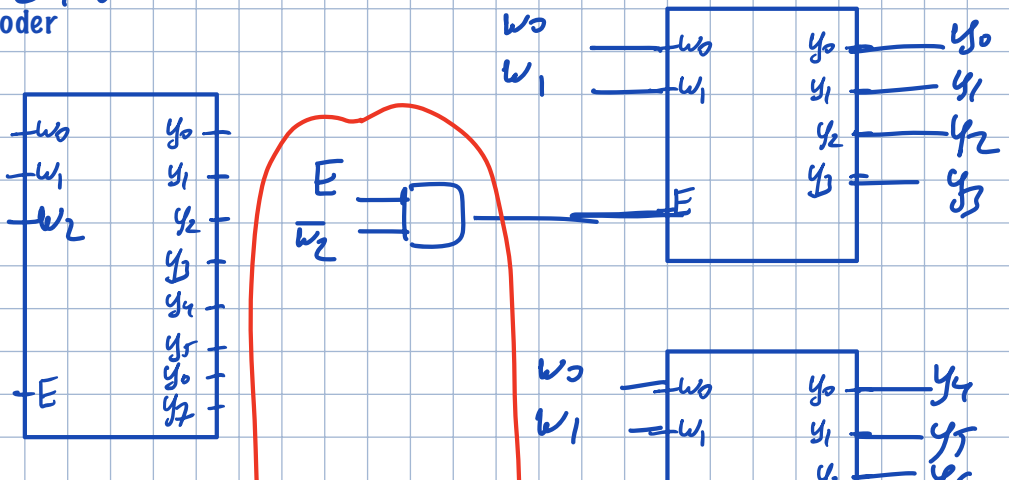
a decoder produces minterms

e.g. $y_0 = m_0 = \bar{w}_1 \bar{w}_0$

$y_0 = E\bar{w}_1\bar{w}_0$
$y_1 = E\bar{w}_1 w_0$
$y_2 = Ew_1\bar{w}_0$
$y_3 = Ew_1 w_0$

4, 3-input AND gates, 2 inverters

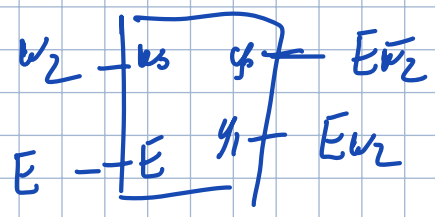
3-to-8 Decoder



→ 1 to 2 decoder with enable

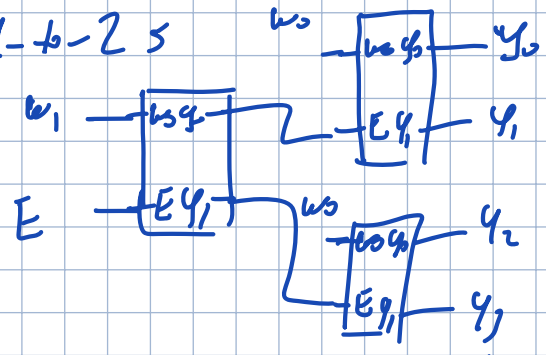
$E w_2 w_1 w_0$	y_7 y_6 -- y_2 y_1 y_0
0 x x x	0 0 -- 0 0 0
1 0 0 0	0 0 -- 0 0 1
1 0 0 1	0 0 -- 0 1 0
⋮	⋮
1 1 1 0	0 1 -- 0 0 0
1 1 1 1	1 0 -- 0 0 0

} are not



2-to-4 from 1-to-2s

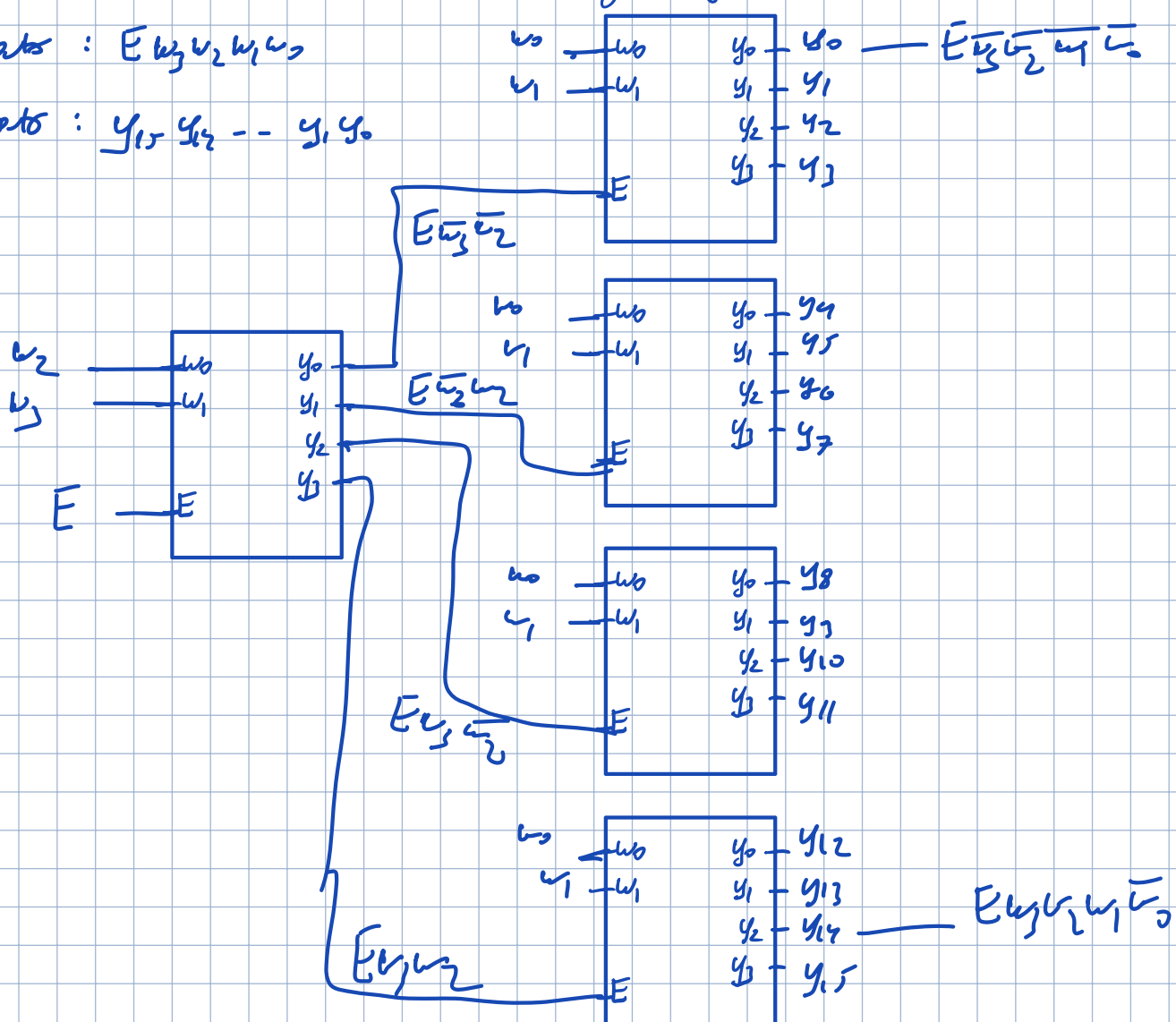
$E\bar{w}_1\bar{w}_0$



Exercise: build 4-to-16 decoder using only 2-to-4 decoders

inputs : $E w_3 w_2 w_1 w_0$

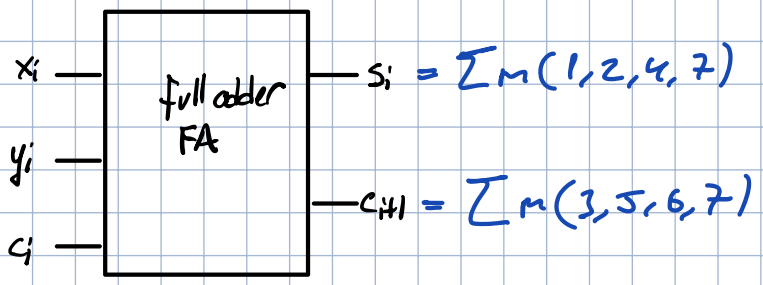
outputs : y_{15} y_{14} -- y_1 y_0

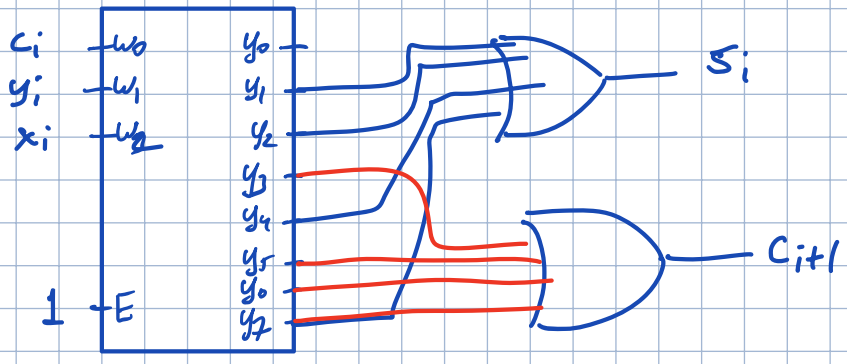


Implementing logic functions using decoders

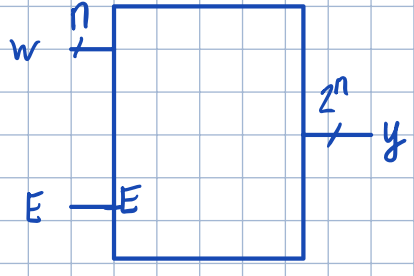
Example: full adder

x_i	y_i	c_i	s_i	c_{i+1}
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





n-to-2^n decoder



* Any n-input logic function can be implemented using n to 2^n decoders and OR gates

Uses of decoders:
memory
I/O
instruction decoding

Demultiplexers

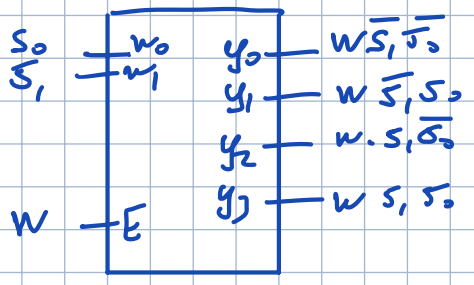
1-to-4 demux

select inputs

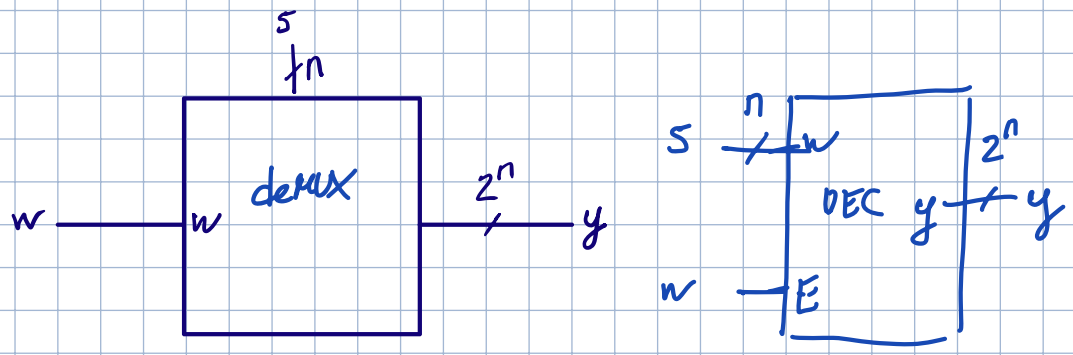
inputs: w, s_1, s_0

s_1	s_0	y_3	y_2	y_1	y_0
0	0	0	0	0	w
0	1	0	0	w	0
1	0	0	w	0	0
1	1	w	0	0	0

2-to-4 decoder with enable



1-to-2^n demultiplexer = n-to-2^n decoder with enable



Encoders

4-to-2 encoder

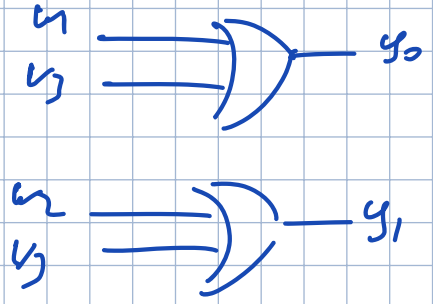
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

one-hot

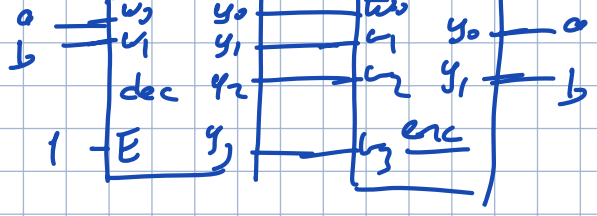
all other inputs never happen

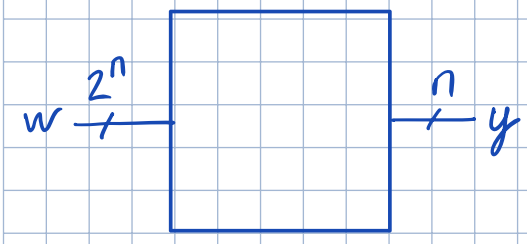
$y_0 = w_1 + w_3$

$y_1 = w_2 + w_3$



2^n to n encoder





Priority encoders

4-to-2 priority encoder

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	X	X	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

$y_0 = w_1 + w_3$?

$y_1 = w_2 + w_3$ ⇒ this does not work

0110 → no longer a don't care
$y_0 = 1$ } violates the truth table

intermediate outputs

w_3	w_2	w_1	w_0	i_3	i_2	i_1	i_0	y_1	y_0	z
0	0	0	0	0	0	0	0	X	X	0
0	0	0	1	0	0	0	1	0	0	1
0	0	1	X	0	0	1	0	0	1	1
0	1	X	X	0	1	0	0	1	0	1
1	X	X	X	1	0	0	0	1	1	1

just 4-to-2 encoder

?

$z = i_0 + i_1 + i_2 + i_3$ or
$w_0 + w_1 + w_2 + w_3$ or
not $y_0 \cdot y_1$

$y_1 = i_2 + i_3$

$y_0 = i_1 + i_3$

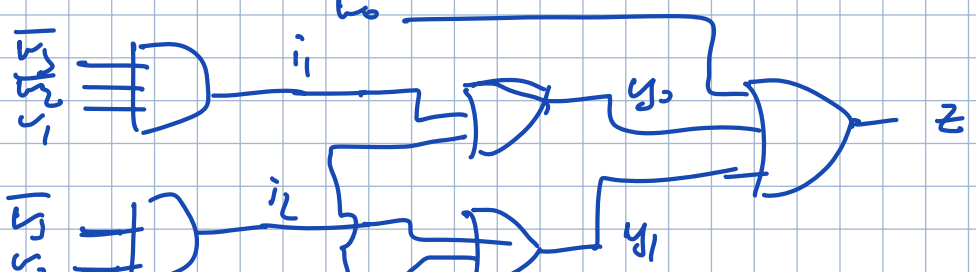
$i_3 = w_3$

$i_2 = \overline{w_3} \cdot w_2$

$i_1 = \overline{w_3}\,\overline{w_2}\,w_1$

~~$i_0 = \overline{w_3}\,\overline{w_2}\,\overline{w_1}\,w_0$~~

no need to implement this one



Comparators (Unsigned) non-negative numbers

4-bit comparator

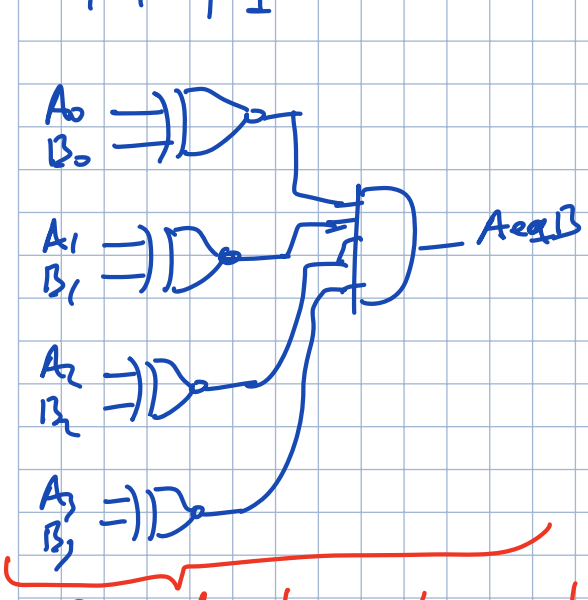
Inputs: $A = A_3A_2A_1A_0$ $B = B_3B_2B_1B_0$

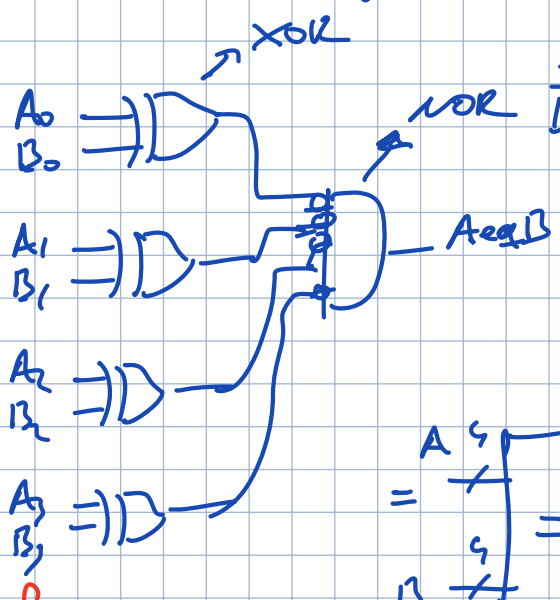
Output: $AeqB = \begin{cases} 1 & \text{if } A = B \\ 0 & \text{if } A \neq B \end{cases}$

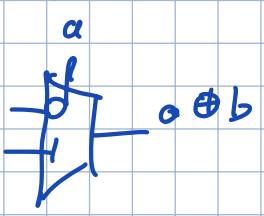
a	b	a XNOR b
0	0	1
0	1	0
1	0	0
1	1	1

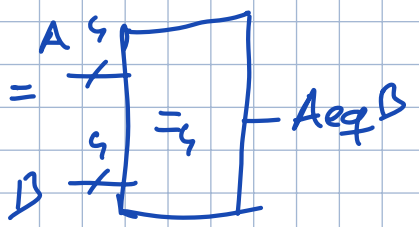
$I_k = A_k \odot B_k$

$AeqB = I_1 \cdot I_2 \cdot I_3 \cdot I_4$ → can be implemented using 2-to-1 MUX







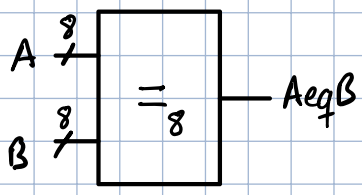


Can also be used to check if two numbers A & B are equal in two's complement representation

8-bit comparator from 4-bit comparator

Inputs: A, B (8 bit)

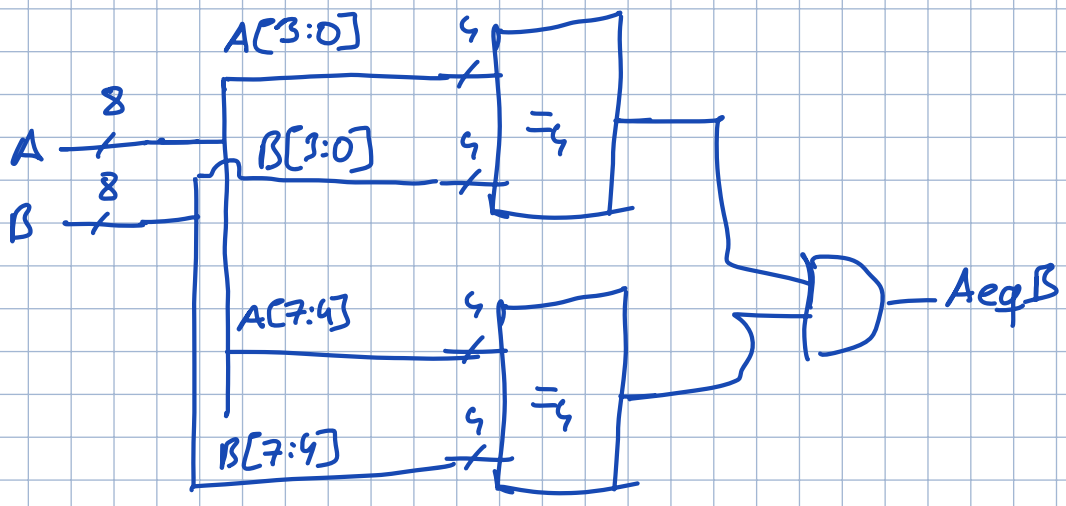
Output: $AeqB = \begin{cases} 1 & A = B \\ 0 & A \neq B \end{cases}$



$AeqB = (AeqB)_{MSN} \cdot (AeqB)_{LSN}$

↑ most significant nibble

↑ least significant nibble



does not work for two's complement

2-bit A>B (unsigned)

Inputs : $A = A_1 A_0$ $B = B_1 B_0$

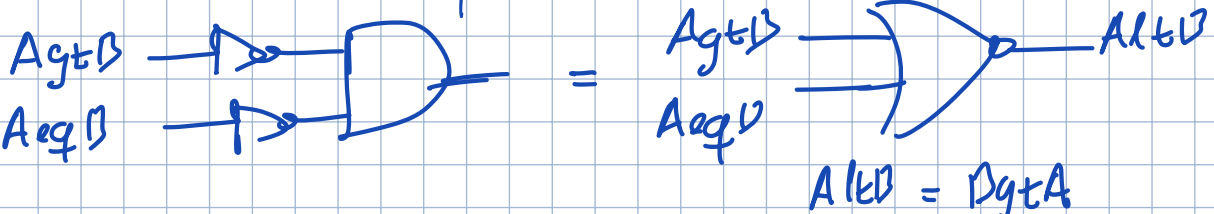
$AgtB_2 = A_1 \bar{B}_1 + (A_1 \odot B_1) A_0 \bar{B}_0$

Output : $AgtB = \begin{matrix} 1 & A > B \\ 0 & A \le B \end{matrix}$

3-bit A>B ; $AgtB_3 = A_2 \bar{B}_2 + (A_2 \odot B_2) A_1 \bar{B}_1 + (A_2 \odot B_2)(A_1 \odot B_1) A_0 \bar{B}_0$

$= A_2 \bar{B}_2 + (A_2 \odot B_2) AgtB_2$

Exercise: 2-bit A<B (output: AltB)

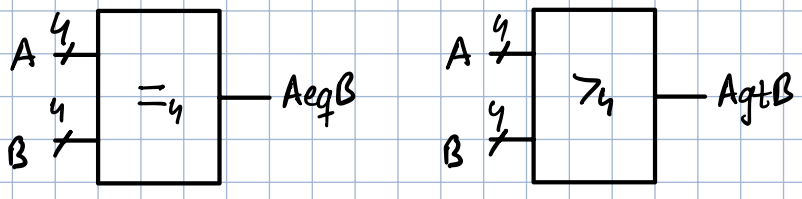


$AltB = BgtA$

8-bit A>B using (A>B)_4 and (A=B)_4

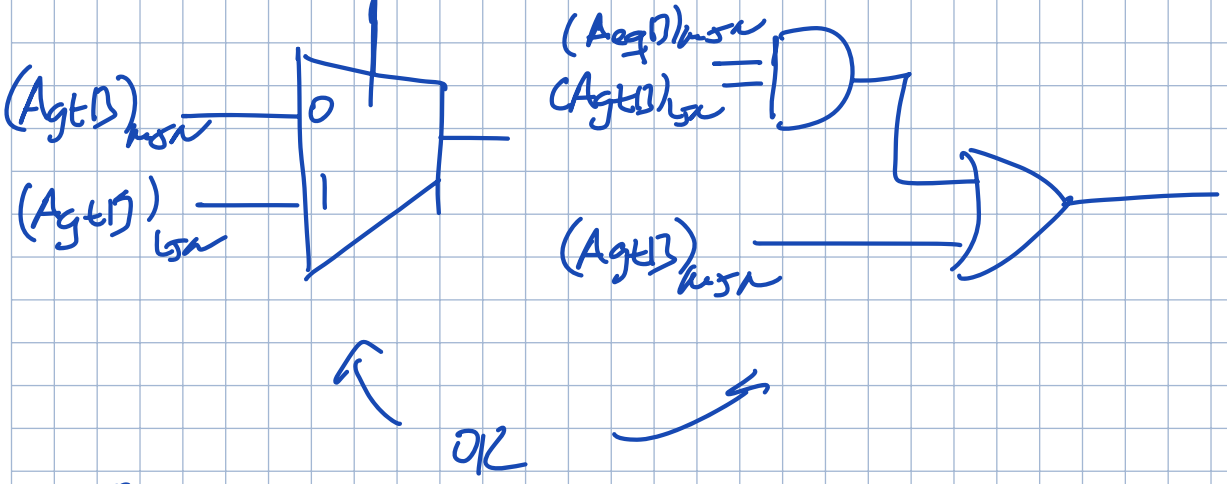
Inputs : A, B (8 bits)

Output : $AgtB = \begin{matrix} 1 & A > B \\ 0 & A \le B \end{matrix}$

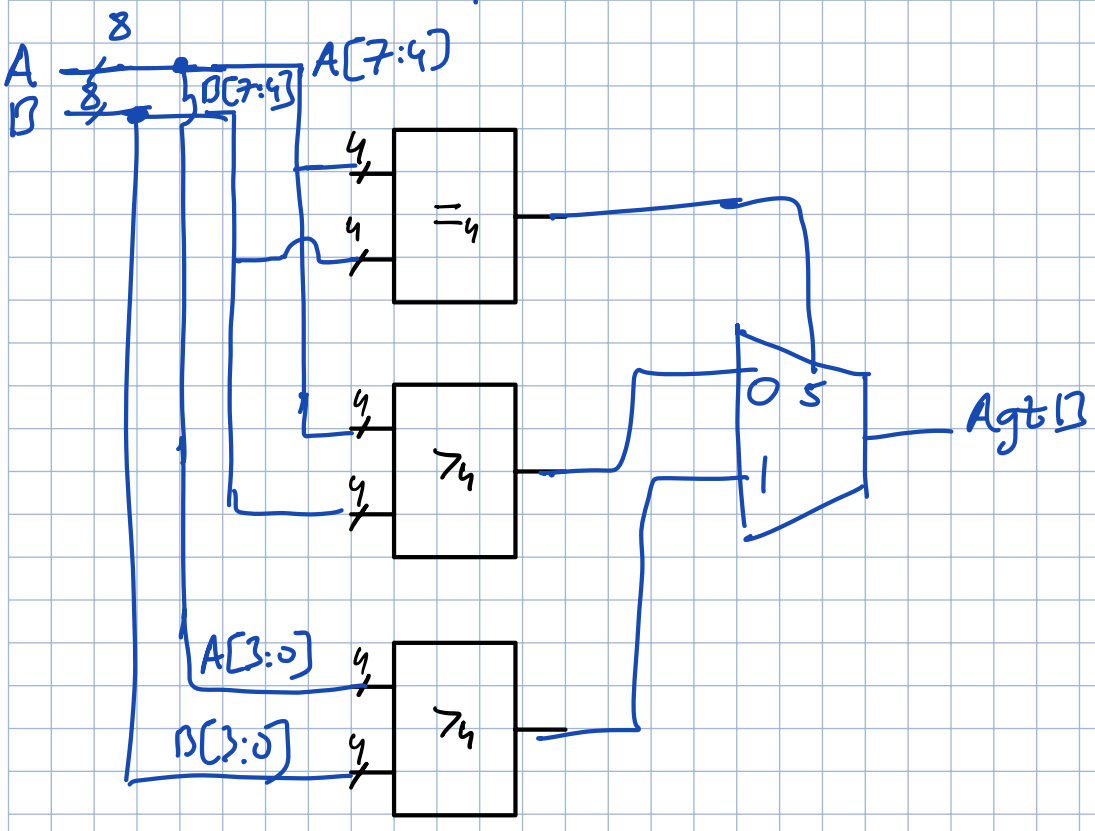


$AgtB = (AgtB)_{MSN} + (AeqB)_{MSN} \cdot (AgtB)_{LSN}$

$(AeqB)_{MSN}$


(Agt[3])msw
(Agt[3])lsw
0
1
(Aeq[3])msw
(Agt[3])lsw
=
(Agt[3])msw

OK


8
A
B
8
A[7:4]
B[7:4]
4
4
=4
4
4
>4
0
S
1
Agt[3]
A[3:0]
B[3:0]
4
4
>4