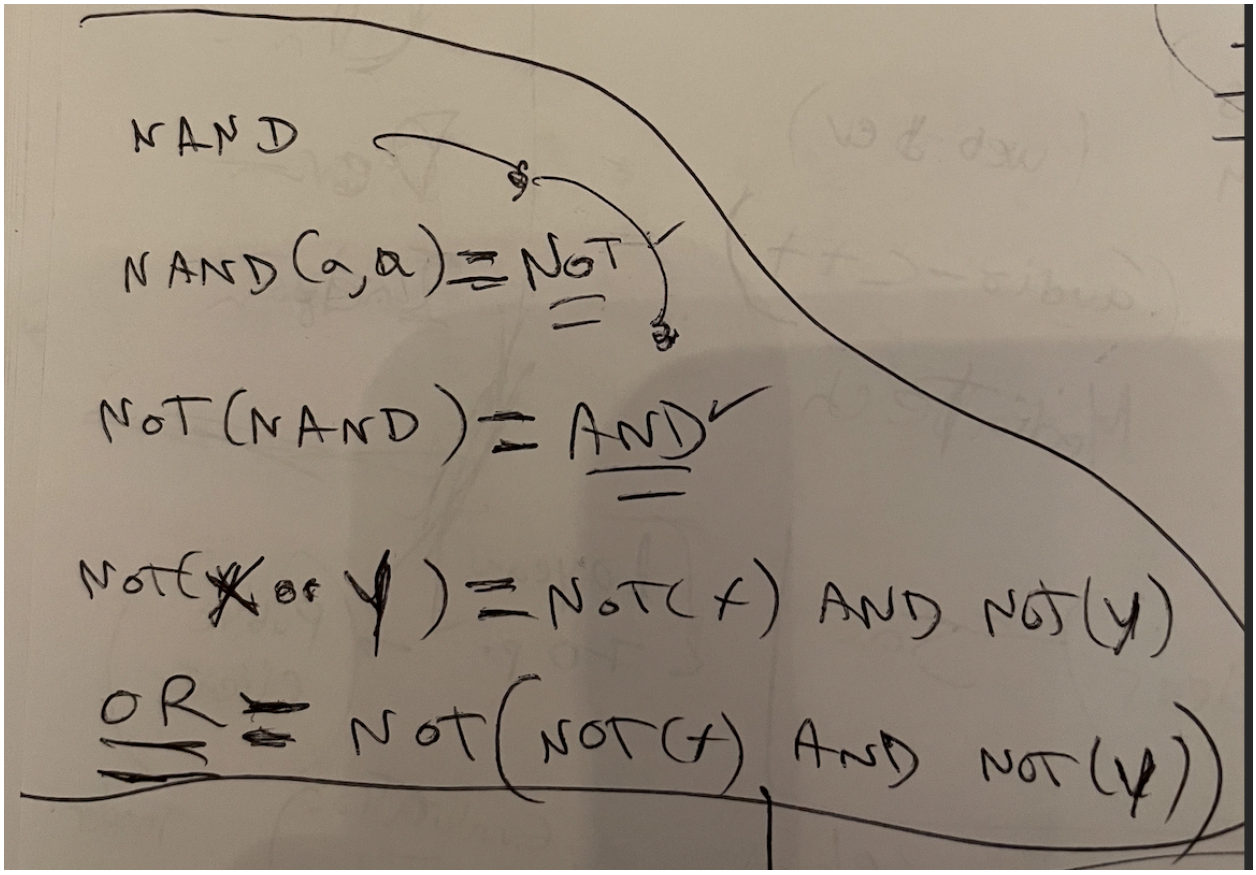


In “Elements of Computing Systems”, NAND is used as the most primitive gate. All other gates are built on top of it. (This is not the only way, it’s *a way,* but other gates could be chosen as the starting point too)

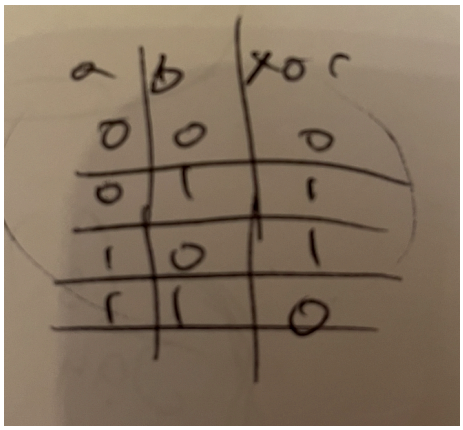
From **Nand**, you first get **Not**. (Simply by giving NAND(a,b) the same variable => Not(a) = NAND(a, a)). This can be checked easily with a truth table.

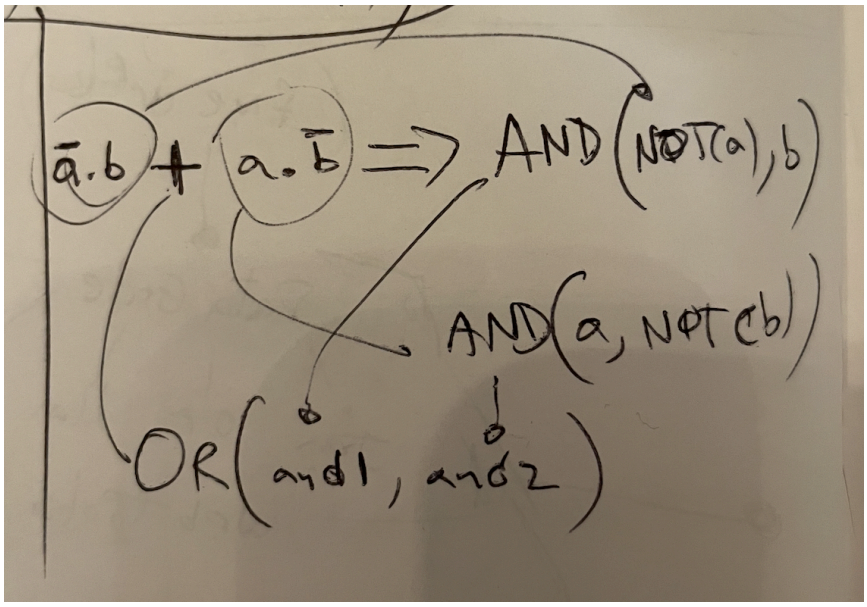
With Nand and Not, you can build an **AND,** by negating NAND.
not(NAND) = AND. This is obvious because by definition NAND is not-and, so the negation just goes back to and.

Now that And and Not are available, the **OR** can be built.
The **OR** comes from a relationship known as *De Moore’s Law:*
Not(X or Y) = Not(X) and Not(Y)

From this, the Or can be defined as the negation of the not in the equation above: OR = Not(Not(X) and Not(Y))

Next, the **Xor** can be created with two ands, two nots, and one or. By using the truth table of the xor, you can obtain this by Boolean algebra of the function from that table:

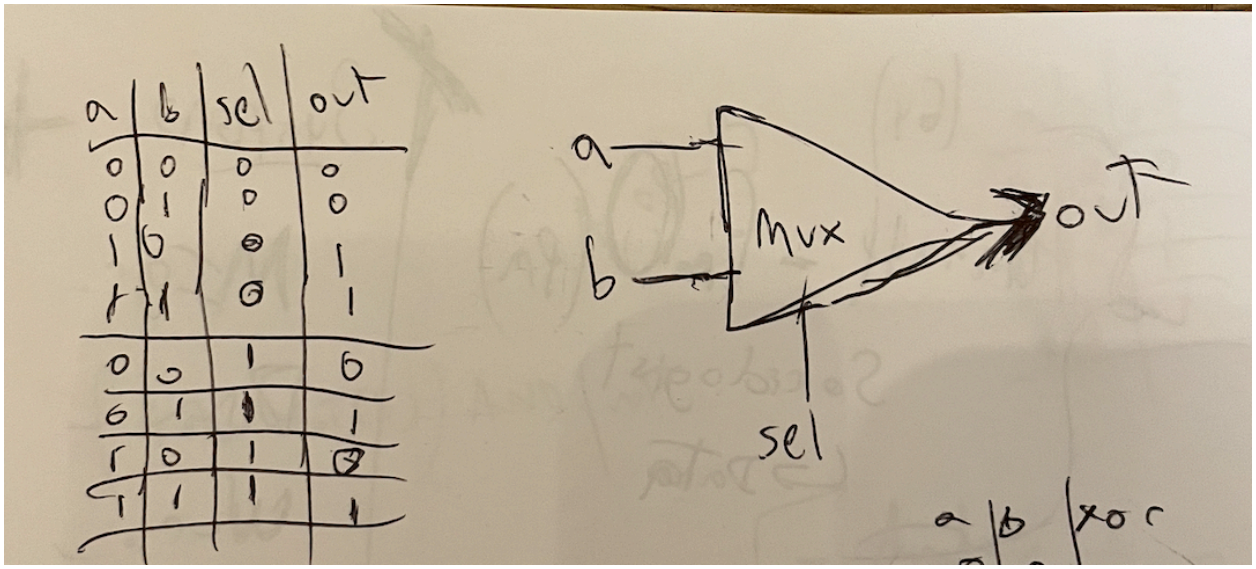


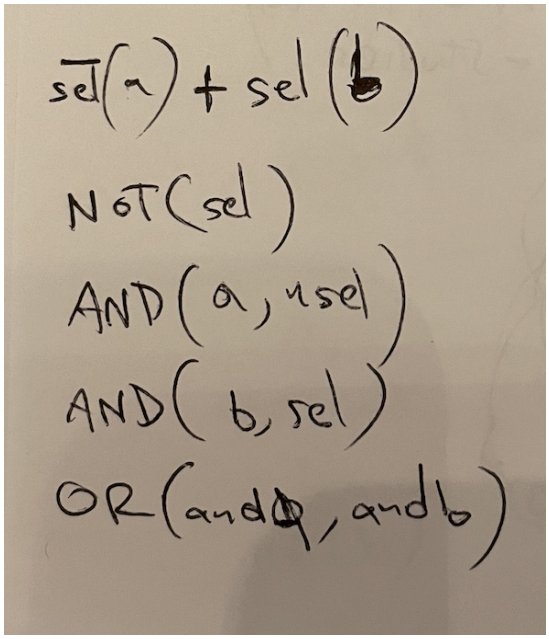


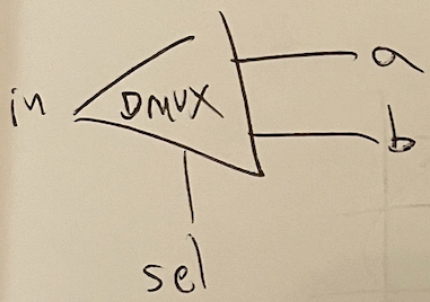
The result: Xor = Or(And(Not(a), b), And(a, Not(b)))

After this, versions for more-than-1-bit gates can be done with buffers, see HDL implementations for And16, Not16, Or16, and Or8way (merge all Ors from a 8-bit variable into one binary output)

Multiplexers and Demultiplexers:







in	sel	a	b
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	0

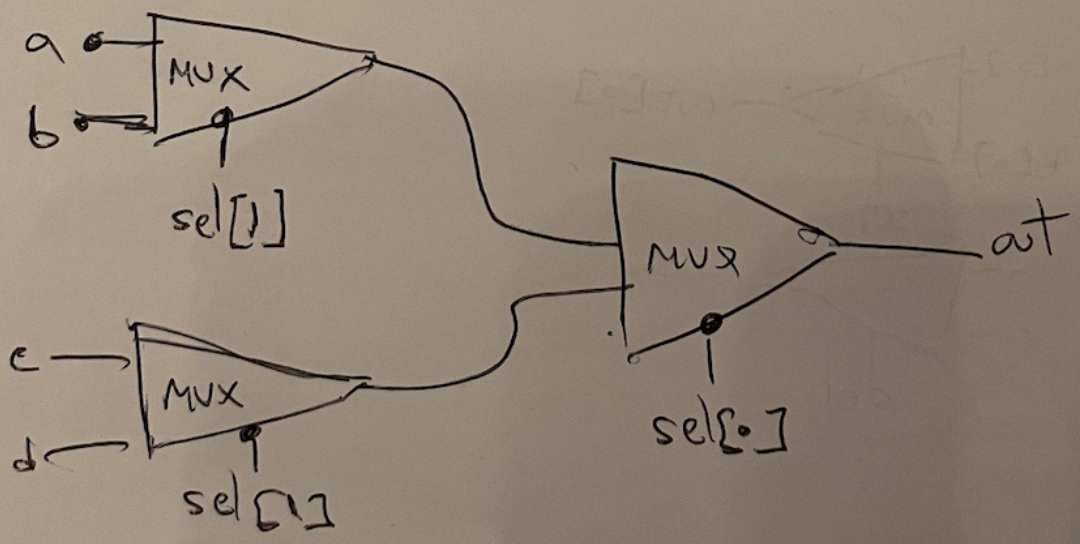
$in.\overline{sel} + in.sel$

Mux4Way16

a	b	c	d	sel[1]	sel[0]	out
a	—	—	—	0	0	a
—	b	—	—	0	1	b
—	—	c	—	1	0	c
—	—	—	d	1	1	d

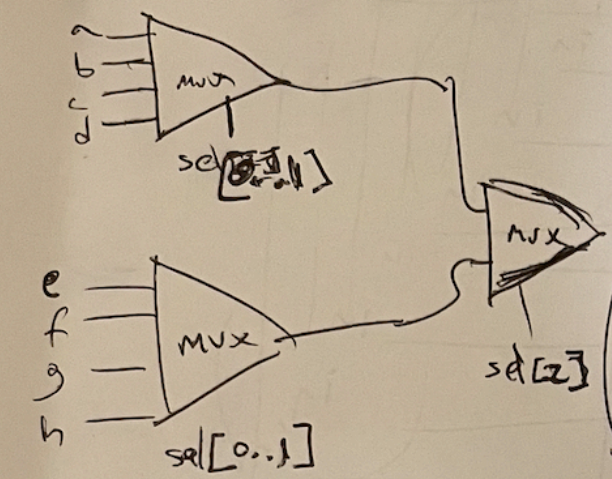
$$\left(\overline{sel[0]} \cdot \overline{sel[1]} \cdot a + \overline{sel[0]} \cdot sel[1] \cdot b\right) + \left(sel[0] \cdot \overline{sel[1]} \cdot c + sel[0] \cdot sel[1] \cdot d\right)$$

$$= \overline{sel[0]} \cdot \left(\overline{sel[1]} \cdot a + sel[1] \cdot b\right) + sel[0] \cdot \left(\overline{sel[1]} \cdot c + sel[1] \cdot d\right)$$



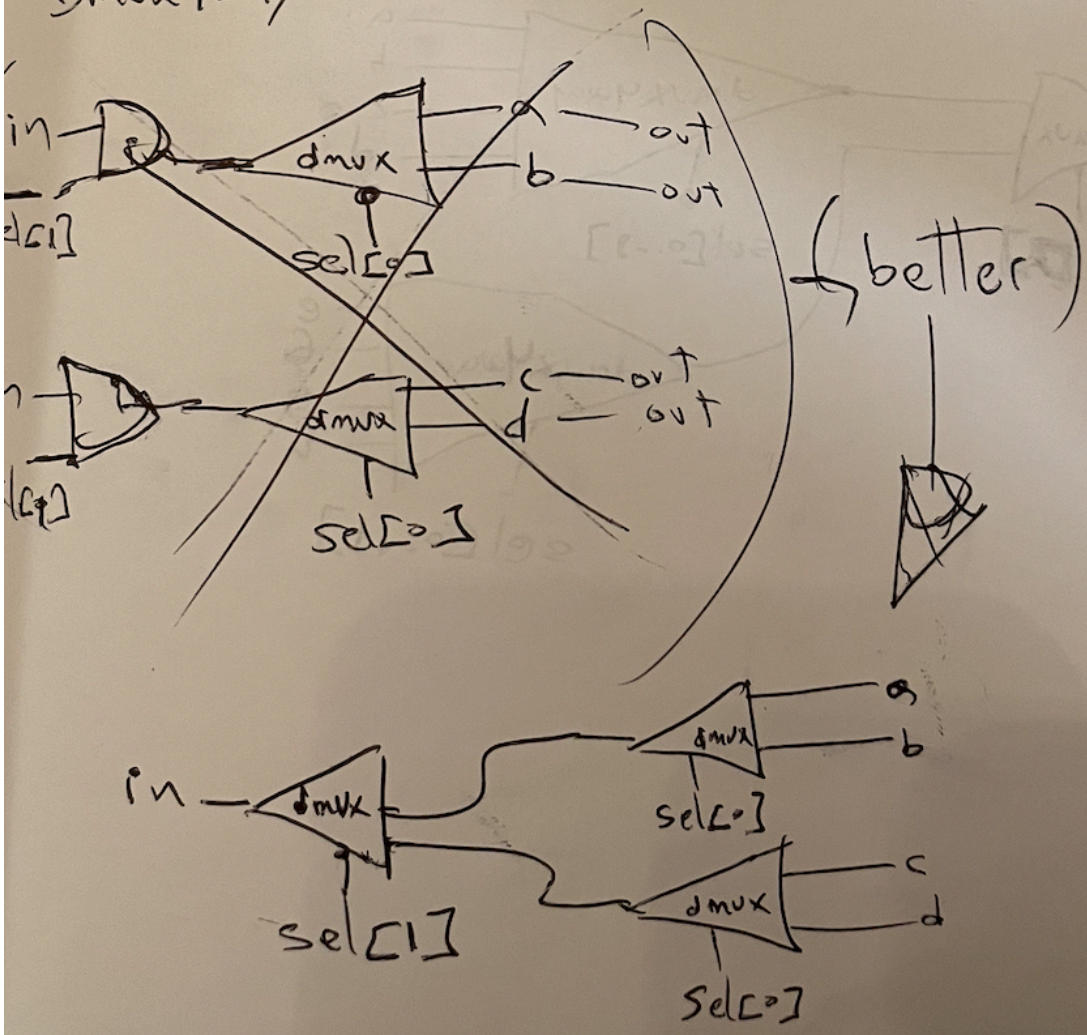
mux 8way 16

a b c d e f g h



sel[2]	sel[1]	sel[0]	out
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

DMux4way



(better)

Dmux8way

in	sel[2]	sel[1]	sel[0]	a	b	c	d	e	f	g	h
	0	0	0	in							
	0	0	1		in						
	0	1	0			in					
	0	1	1				in				
	1	0	0					in			
	1	0	1						in		
	1	1	0							in	
	1	1	1								in