Module 1 - Boolean and Bits

Theorem

All boolean functions can be represented using an expression containing AND, OR and NOT.

All boolean functions can be represented using an expression containing AND, and NOT.

All boolean functions can be represented using an expression containing NAND, where

$$\text{NAND}(x, y) = \text{NOT}(x \text{ AND } y).$$

Proof

1. Won't prove the first one.
2. Second one is by de Morgan's law:

$$x \text{ OR } y = \text{NOT}(\text{NOT}(x) \text{ AND } \text{NOT}(y)).$$

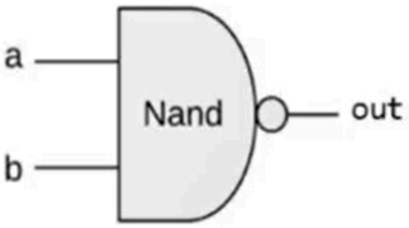
3. Third one is because:
    1. $\text{NOT}(x) = \text{NOT}(x \text{ AND } x) = \text{NAND}(x, x).$
    2. $\text{AND}(x) = \text{NOT}(x \text{ NAND } x).$

Logic Gates

- Elementary (e.g. OR, NOT, AND, NAND, ...)
- Composite: made up of elementary logic gates (e.g. Mux, Adder, ...)

NAND Gate

gate diagram:



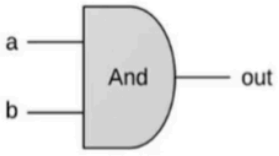
functional specification:

```
if (a==1 and b==1)
then out=0 else out=1
```

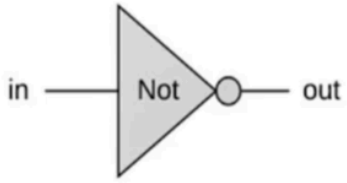
truth table:

a	b	out
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

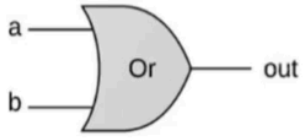
AND, OR, NOT Gate




```
if (a==1 and b==1)
then out=1 else out=0
```

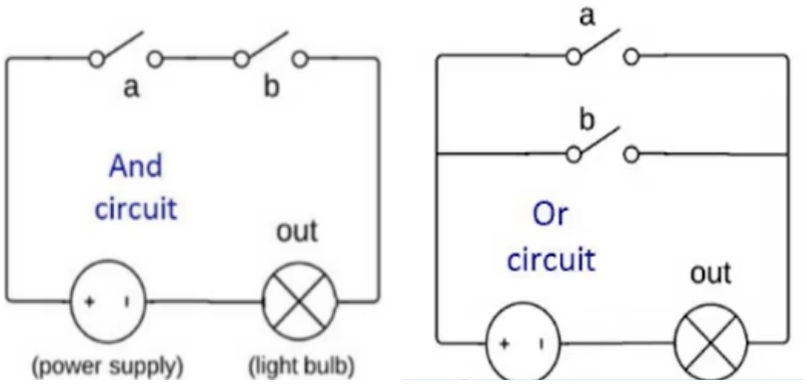



```
if (in==0)
then out=1 else out=0
```



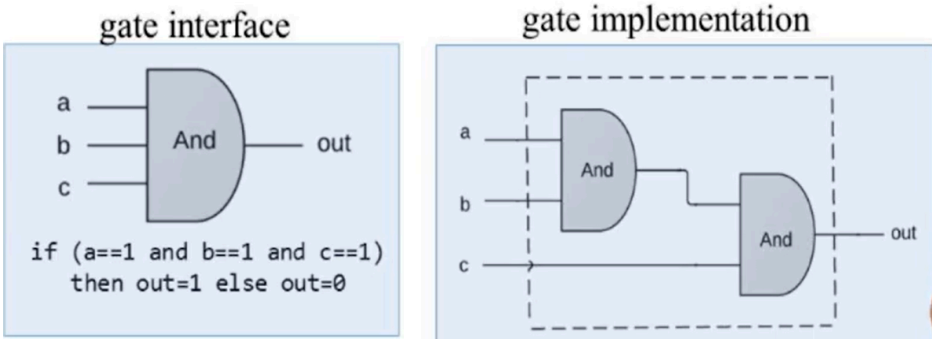

```
if (a==1 or b==1)
then out=1 else out=0
```

Circuit Implementation

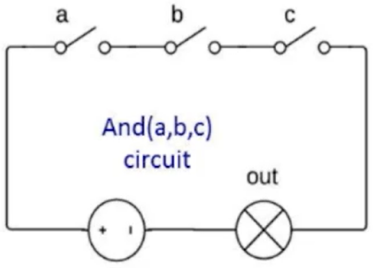


Composite Gates

- Three-way AND gate



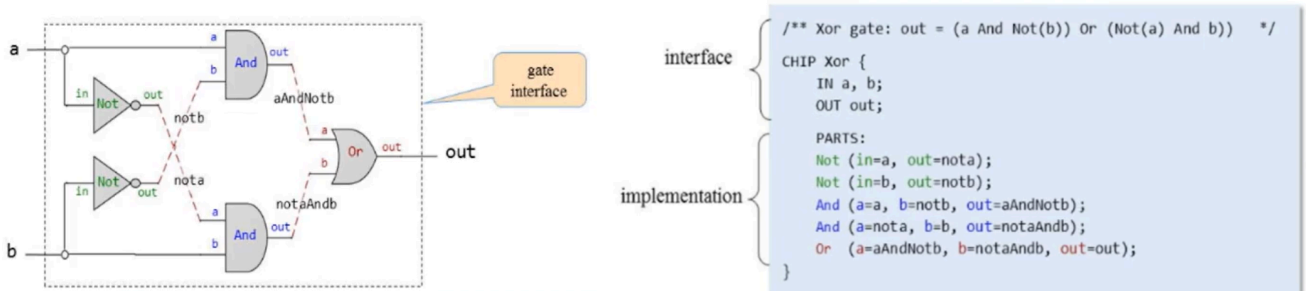
Circuit Implementation (Three-way AND)



HDL (Hardware Description Language)

- HDL is a functional/declarative language
- The order of HDL statements is insignificant
- Before using a chip part, we must know its interface.

Example: Xor gate




```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b) */

CHIP Xor {
    IN a,b;
    OUT out;

    PARTS:
    Not (in=a, out=notb);
    Not (in=b, out=nota);
    And (a=a, b=notb, out=aAndNotb);
    And (a=nota, b=b, out=notaAndb);
    Or (a=aAndNotb, b=notaAndb, out=out);
}
```

Multi-bit buses

- Sometimes we want to manipulate a bunch of bits together.
- Conceptually, it is convenient to think of such a group of bits as a single entity, sometime termed "bus".
- HDLs will usually provide convenient notation for handling these buses
- Multi-bit buses are indexed from right to left, and indexed from 0.
- Example: Add16 adds 16 bits numbers together

```
CHIP Add16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
        ...
}
```