Logic gates

Logic gates are the basic building blocks of digital computing. **A logic gate is an electrical circuit that has one or more than one input and only one output.** The input controls the output and is isomorphic with logical conditions that can be expressed in the form of truth-tables.

Truth tables

I know from my study of logic that truth tables enable us to present the conditions under which logical propositions are true or false. To take the `AND` operator: `AND` evaluates to `true` if both of its constituent expressions are `true` and `false` in any other circumstances (e.g. if one proposition is `true` and the other `false` (or vice versa) and if both propositions are `false`).

This is most clearly expressed in the following truth table:

Truth table for `AND`

```
p    q   p & q
-    -   -----
t    t     t
t    f     f
f    t     f
f    f     f
```

Another example is the negation (`NOT`) operator in logic which is highly trivial. The negation operator (¬ or `~`) switches the value of a proposition from true to false. When we put `~` before `true` it becomes false and when we put `~` before `false` it becomes `true`. We will see shortly that this corresponds to a basic on/off switch.

Truth table fo `NOT`

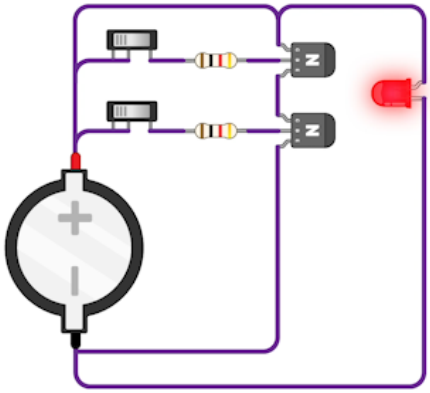
```
p    ~ p
-    --
t    f
f    t
```

NAND gates

A NAND gate is a logic gate that combines the truth conditions for `AND` and `NOT` . I

Let's first introduce the circuit:

The real-life circuit showing two switches corresponding to two transistors which



control the LED light.

In this circuit, there are two transistors, each connected to a switch. The switches control the LED light. So the switches are the input and the LED is the output.

For clarity, we are not going to draw both transistors, we will simplify the diagram with a symbol for them which stands for the NAND gate:

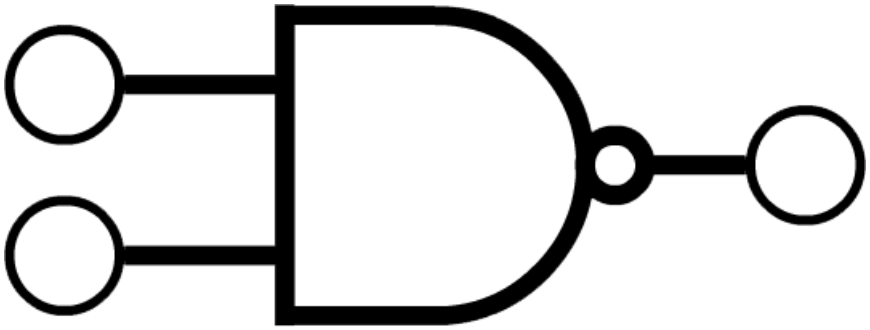


Figure 1: NAND.png

> Remember that a 'logic gate' is a logical abstraction of a physical process: the voltage passing through a transistor. The transistors register the charge and the switches control it's flow, the 'gate' is just the combination of transistors and how they are arranged. There is not a physical gate per se, there is only the transistor whose output we characterize in terms of logic.

The diagram below shows how the circuit models the truth conditions for `AND`

Diagram representing NAND gate:

- When both switches are off (corresponding to `false false`) the output is

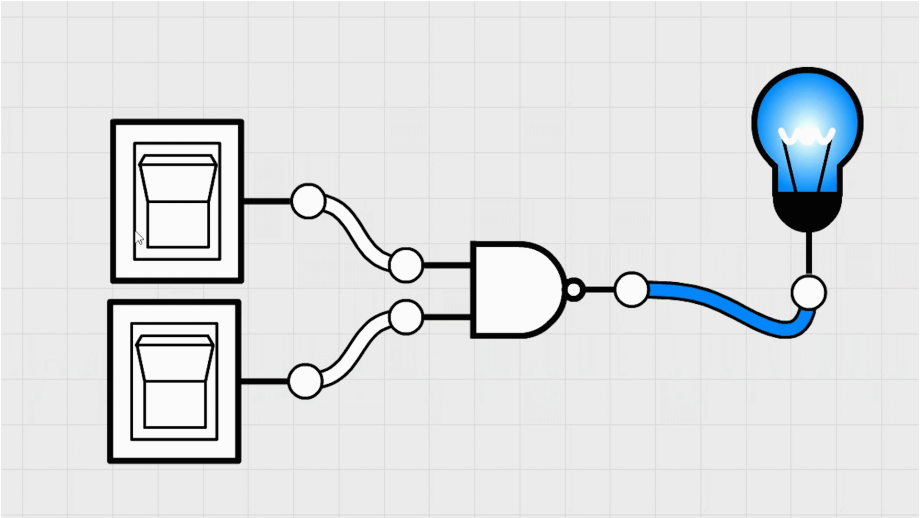


Figure 2: NAND.gif

on (the bulb lights up).

- If either one of the switches are on, the output remains on (corresponding to `true false` or `false true`)
- It is only when both switches are on, that the output is off (corresponding to `true true`)

  Remember that switch circuitry is counter intuitive: the switches being on corresponds to the output ceasing to execute because the switches break the circuit, they don't join it.

Transliterating the logic truth table to the switch behaviour

We can now present a truth table for NAND alongside the truth conditions for `AND` and `NOT`

```
// AND
p    q   p & q
-    -   -----

t    t     t      (1)
t    f     f      (2)
f    t     f      (3)
f    f     f      (4)

// NOT

p    ~ p
```

```
-    --
t    f
f    t

A    B   Output

-    -   -----

0    0     1       (1)
1    0     1       (2)
0    1     1       (3)
1    1     0       (4)
```

- So we can see that the binary representation of the circuit accords with `NOT` at rows (1) and (4): when both switches are off (`false`), the bulb is on (`true`). And when both switches are on (`true`), the bulb is off (`false`).
- Rows (2) and (3) of the binary truth table accord with rows (2) and (3) of the `AND` truth table: if one of the switches is `true` but the other is `false` , the output is `false` (the bulb remains on).

More complex outputs from combining NANDS

The example we have looked at so far is fairly simple because there is just one NAND gate corresponding to two inputs (the two switches) and one output (the bulb).

When we add more NAND gates and combine them with each other in different ways we can create more complex output sequences and these two will have corresponding truth tables.

`NOT` gate

This gate corresponds to the `NOT` Boolean or negation logical connective. It is really simple and derived from the trivial logical fact that `true` is `true` and `false` is `false` also known as **logical identity**.

Natural language

> The negation operator (¬ or `~`) switches the value of a proposition from `true` to `false`. When we put `~` before `true` it becomes `false` and when we put `~` before `false` it becomes `true` .

Truth table

This corresponds to a simple on-off switch.

In terms of logic gates we would create this by using a single NAND gate. Although it can take a total of two inputs, it would be controlled by a single switch, so both inputs would be set to `1 1` or `0 0` when the switch is activated

p	T
T	T
F	T

or

p	T
1	1
0	1

Figure 3: 1-w2ILS6M9pgmLcK6V1PEs3Q.png

and deactivated. This would remove the `AND` aspect of `NAND` and reduce it to `NOT` .

A NAND gate simulating NOT logic

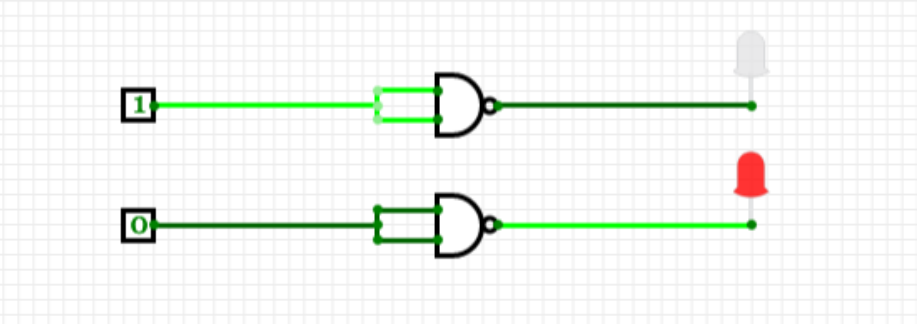


Figure 4: Screenshot_2020-08-25_at_15.09.01.png

Symbol for `NOT` gate

NOT has its own electrical signal to distinguish it from a NAND:

`AND` gate

Just as we can create `NOT` logic from a NAND gate, without the `AND` conditions, we can create a circuit that exemplifies the truth conditions of `AND` without including those of `NOT`.

When we attach two NAND gates in sequence connected to two switches as input this creates the following binary conditions:

```
A    B   Output
-    -   -----
```

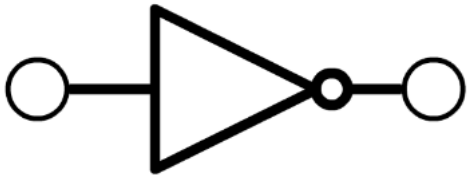


Figure 5: Screenshot_2020-08-25_at_15.18.34.png

```
0    0     0       (1)
1    0     0       (2)
0    1     0       (3)
1    1     1       (4)
```

Which is identical to the truth table for `AND` :

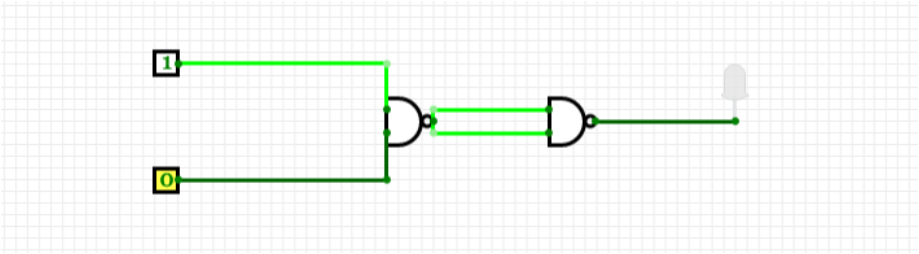
```
p    q   p & q
-    -   -----

t    t     t      (1)
t    f     f      (2)
f    t     f      (3)
f    f     f      (4)
```

Natural language

> `AND` (`&`) is `true` when both constituent propositions are `true` and `false` in all other circumstances viz. `false false` (`¬P & ¬Q / 0 0`), `true false` (`P & ¬Q / 1 0`), `false true` (`¬P & Q / 0 1`)

AND at 0 0

Screenshot_2020-08-25_at_15.04.10 1.png



AND at 1 0 or 0 1

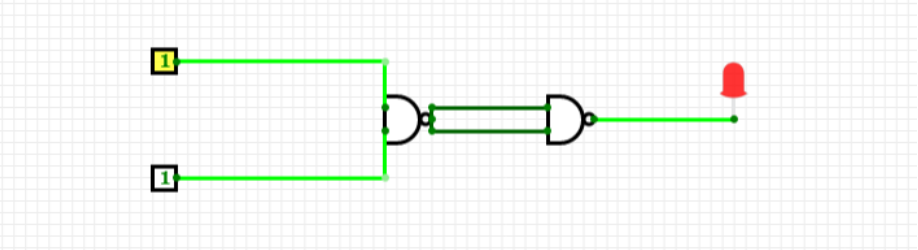


Figure 6: Screenshot_2020-08-25_at_15.05.36.png

Symbol for `AND` gate

It's very similar to NAND so be careful not to confuse it

Pasted image 20220319173651.png

`OR`

> `OR` (in logic known as **disjunction**) in its non-exclusive form is `true` if either of its propositions are `true` or both are `true` . It is `false` otherwise.

Pasted image 20220319173819.png

```
p    q   p V q
-    -   -----

t    t     t      (1)
t    f     t      (2)
f    t     t      (3)
f    f     f      (4)
```

`XOR`

> `XOR` stands for **exclusive or**, also known as **exclusive conjunction**. This means it can only be `true` if one of its propositions are `true` . If both are `true` this doesn't exclude one of the propositions so the overall statement has to be `false` . This is the only change in the truth conditions from `OR` .

Pasted image 20220319173834.png

Electrical symbol for XOR

```
p    q   p X V q
-    -   --------

t    t     f      (1)
```

```
t    f     t      (2)
f    t     t      (3)
f    f     f      (4)
```

NOR

> This is equivalent to saying 'neither' in natural language. It is only `true` both propositions are `false` . If either one of the propositions is `true` the outcome is `false` . If both are `true` it is `false`

Pasted image 20220319173900.png

XNOR

> This one is confusing. I can see the truth conditions but don't understand them. It is `true` if both propositions are `false` like `NOR` or if both propositions are `true` and `false` otherwise.

```
p    q   p ¬V q
-    -   --------

t    t     f      (1)
t    f     f      (2)
f    t     f      (3)
f    f     t      (4)
```

```
p    q   p X¬V q
-    -   --------

t    t     t      (1)
t    f     f      (2)
f    t     f      (3)
f    f     t      (4)
```