LOGIC GATES

Kira Clements, University of Bristol
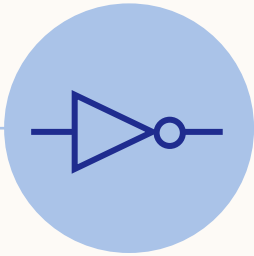
LOGIC GATES

Each logic gate implements a simple boolean function e.g. A ∧ B.

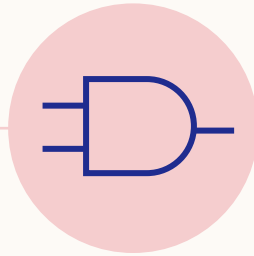These are an abstract view of logical functions!

For now, we'll taking a black box view of these and focus on utilising their functionality rather than looking into *how* they function.
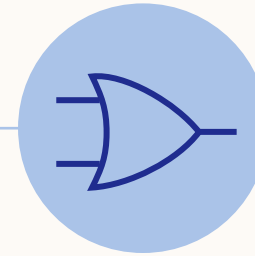
GATE SYMBOLS
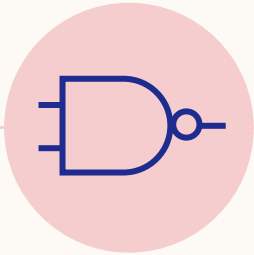
NOT

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

AND

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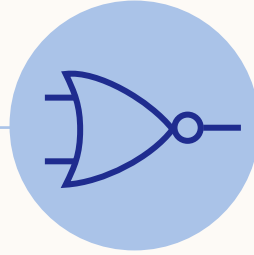then out == 1,
else out == 0

OR

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

MORE GATE SYMBOLS

NAND (NOT-AND)

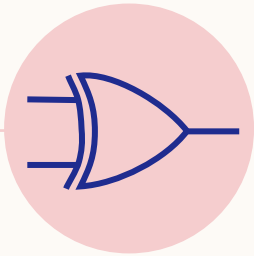if ((a == 1) and (b == 1))
then out == 0,
else out == 1

NOR (NOT-OR)

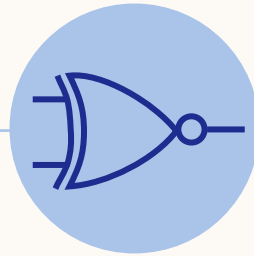if ((a == 0) and (b == 0))
then out == 1,
else out == 0

These are important gates, as each is **functionally complete** i.e every boolean expression can be implemented just using one of these types of gates, making them excellent *building blocks.*

MORE GATE SYMBOLS

XOR

if (a == b) then out == 0,
else out == 1

XNOR (NOT-XOR)

if (a == b) then out == 1,
else out == 0

These gates aren't used as commonly, as they're easily implemented using some combination of the previous logic gates, but it's still useful to be recognise them!
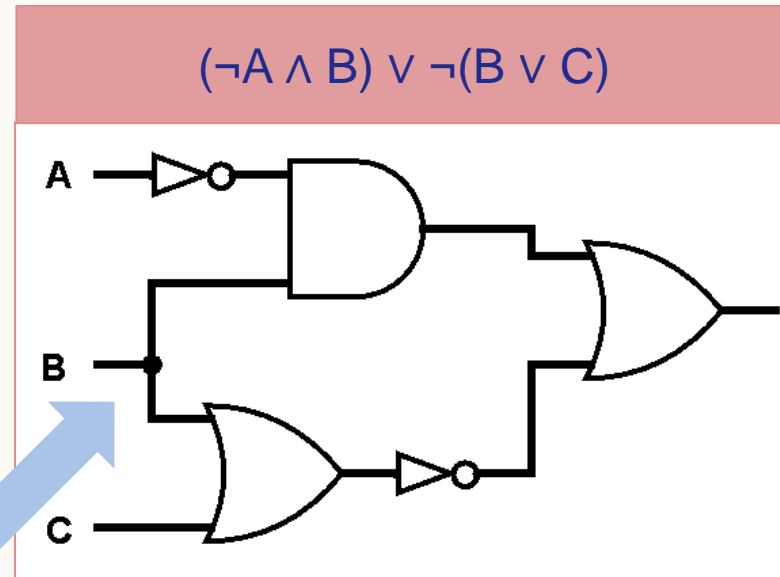
IMPLEMENTING LOGIC

A logical expression can be implemented by constructing a circuit whereby each logical operator is represented by its corresponding gate.

¬A ∧ B

¬(B ∨ C)
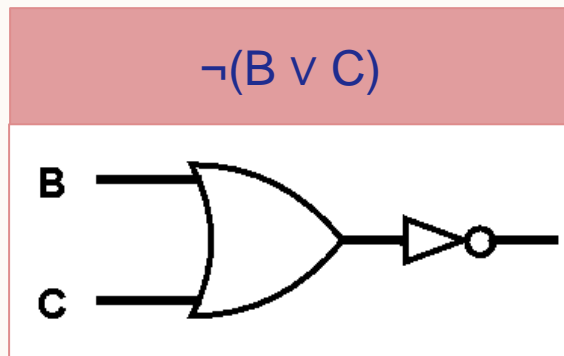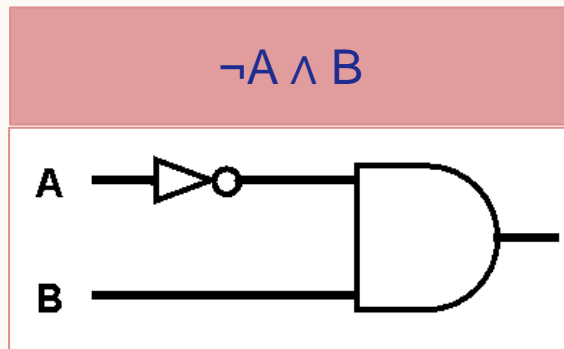
Complex circuits can be built up from simpler ones

Signals can be copied by branching them
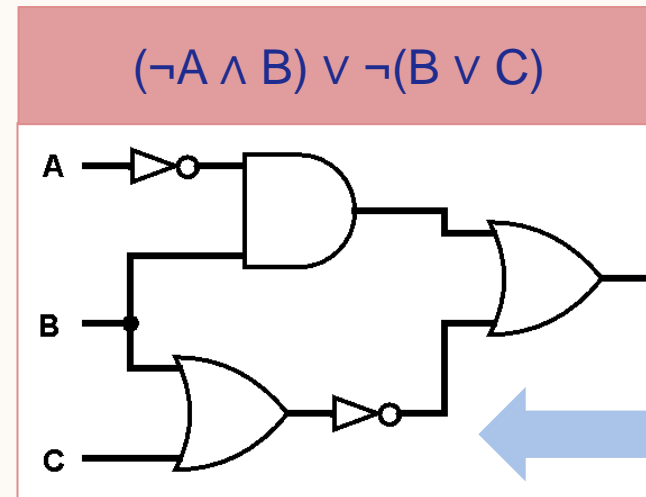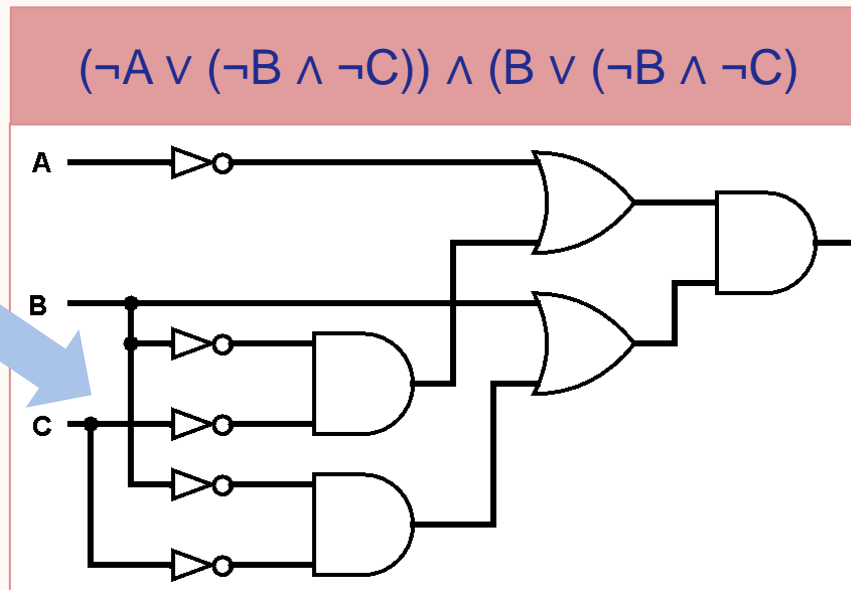
(¬A ∧ B) ∨ ¬(B ∨ C)

IMPLEMENTING LOGIC

A logical expression can be implemented in a variety of ways, but we aim to optimise this by using the least number of gates.

Less gates = lower energy consumption and faster computation!

Rather than using 3 more gates, the output signal from (¬B ∧ ¬C) could be branched

(¬A ∨ (¬B ∧ ¬C)) ∧ (B ∨ (¬B ∧ ¬C)

(¬A ∧ B) ∨ ¬(B ∨ C)

¬(B ∨ C) uses one less gate than (¬B ∧ ¬C)

Both circuits produce the same output, but doesn't the left look a lot harder to understand?
Not only is the basis formula not fully simplified, but there are duplicate gates serving the same purpose.

LOGISIM

Simulation tools are used to test circuit designs before physical implementation (which can be very expensive when produced at scale).

On this course, we'll be using Logisim to design and validate circuits that implement logical expressions. This is an educational tool aiming to help you explore how circuits work.

The full Logisim user guide can be found here.
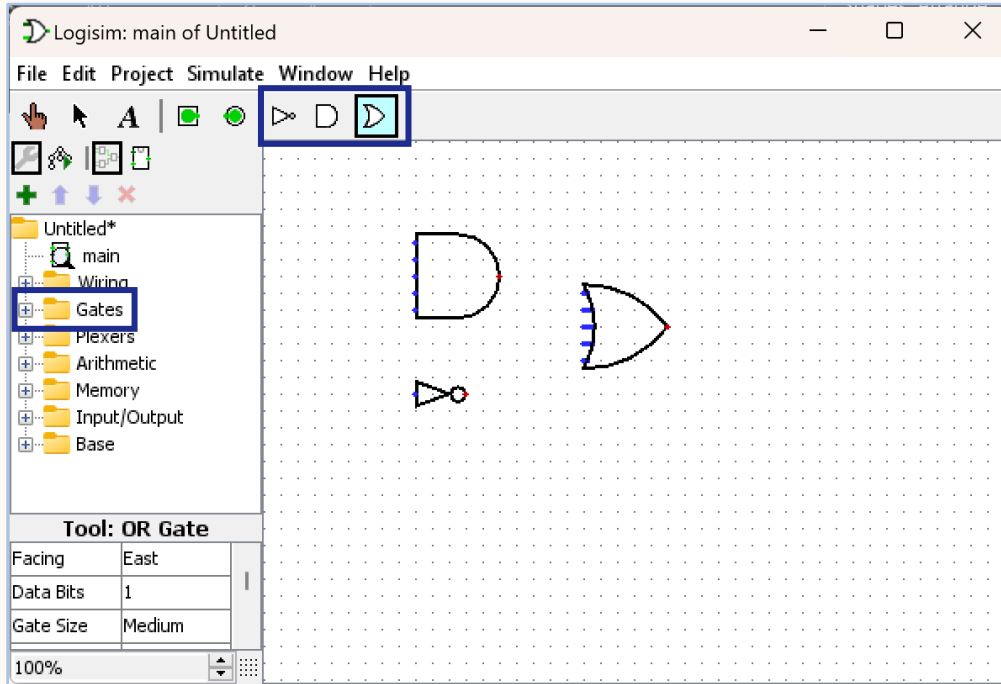
Appropriate **logic gates** can be inserted first, to create a skeleton of your circuit. These can be found under the Gates folder in the explorer pane on the left of the screen, while basic gates are also conveniently located on the toolbar at the top of the screen.
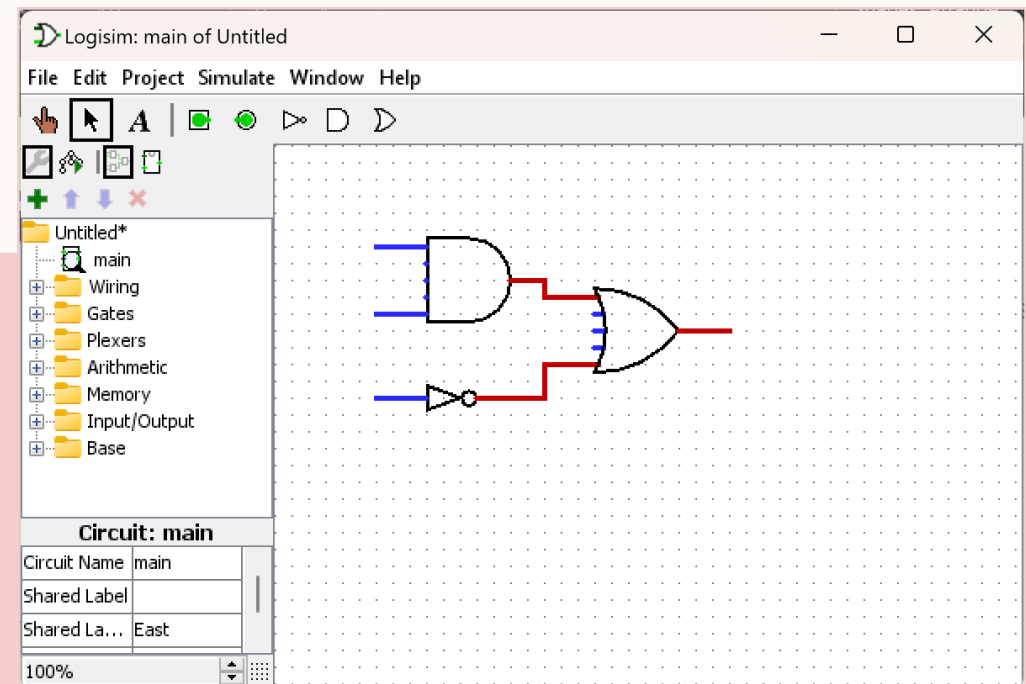
Wires can be dragged using the edit tool ( ) from one spot on a component to link another. Existing wires can be dragged to extend, shorten, or branch them.
Blue represents an unknown 1-bit value.
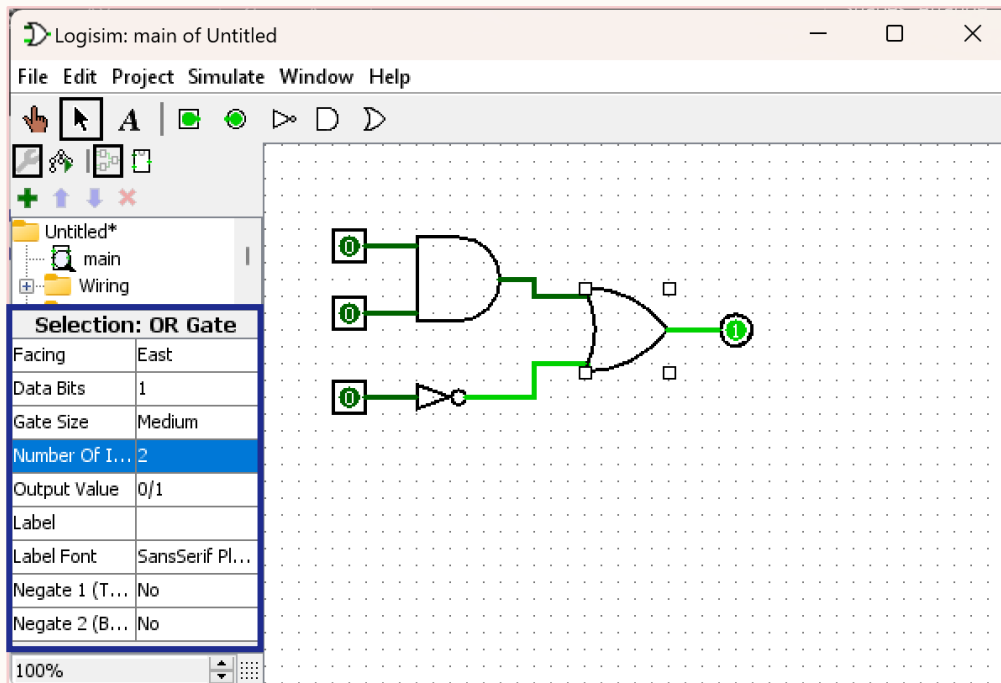**Dark green** represents a 0 1-bit value.
Light green represents a 1 1-bit value.
Red represents an error.

Input (square) and output (round) **pins** are also conveniently located on the toolbar at the top of the screen. These can be connected to the suitable wires in your circuit.

The poke tool (👆) can then be used to toggle the values of the input pins and test whether your circuit outputs what is expected.
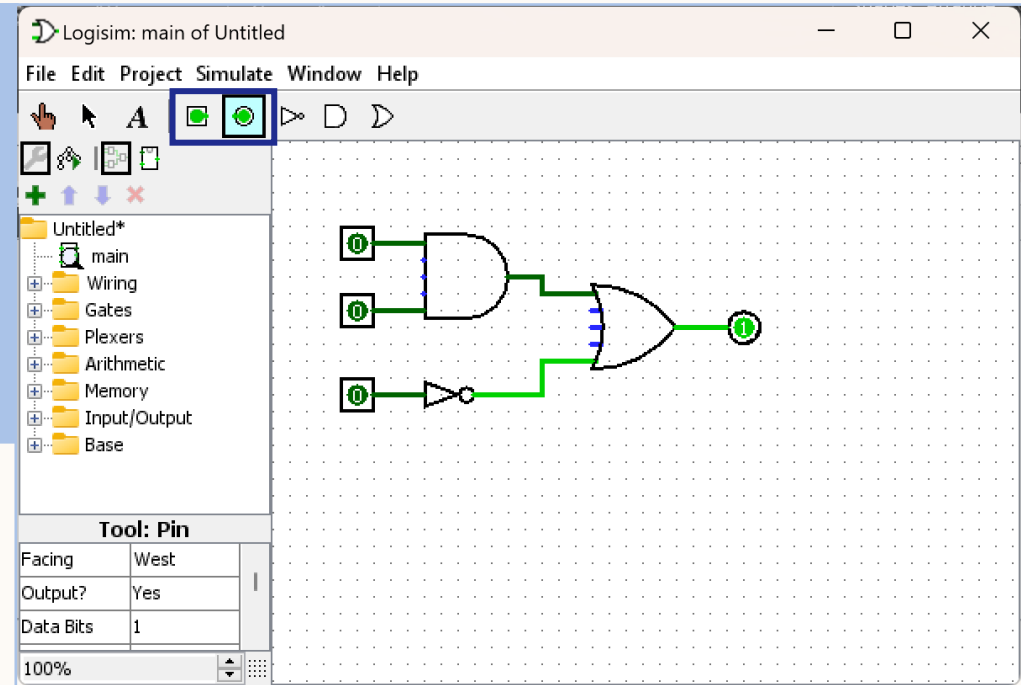
The **attributes** table, located on the left of the screen, allows us to edit components configuration. For example, we can change the number of inputs a logic gates expects, add a label to a component, alter the direction of a component, or change a pin to be an input or output.
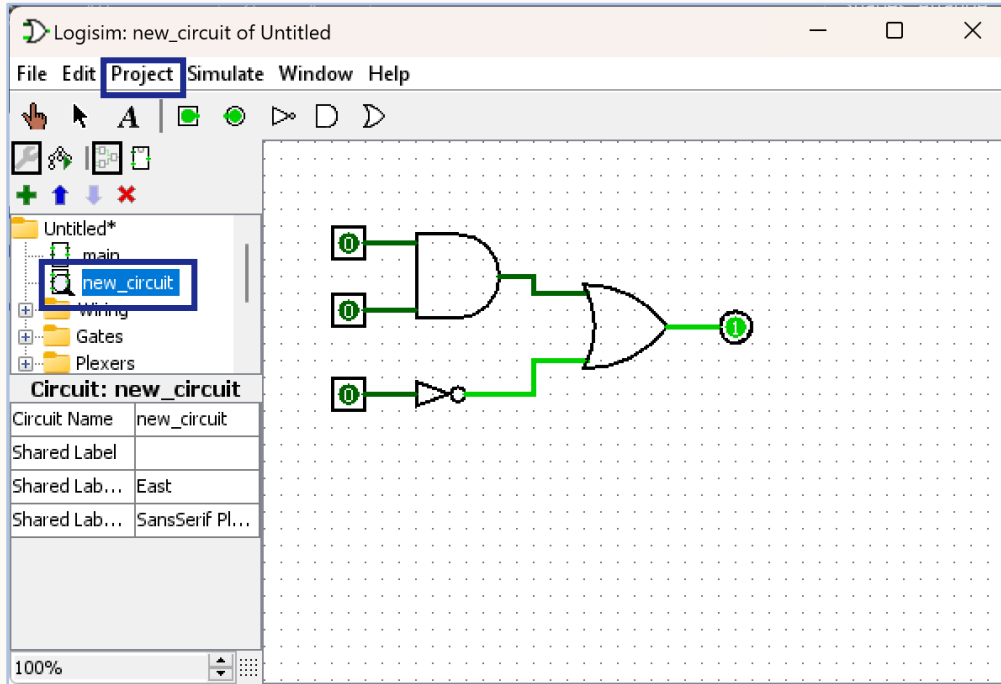
New circuits can be created by clicking on Project > Add Circuit…
Your circuits can then be used like any premade component, by single clicking on it in the explorer pane to use it as a tool. Double-clicking on it in the explorer pane allows you to edit the circuit.

The **appearance** of your newly created component can be edited such as rearranging the input/output pins.
Clicking on a pin will give you a helpful pop up that indicates which pin or your circuit it corresponds to.