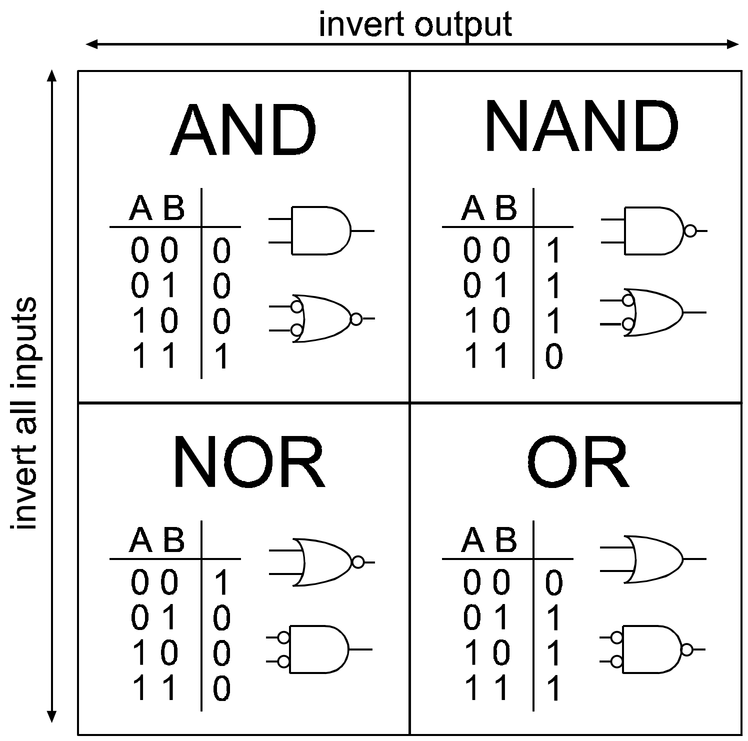
Mixed Logic

Mixed logic is a gate-level design methodology that is widely used in industry. It allows a designer to separate *what* behavior is desired from *how* it is realized. It also supports self-documenting gate level circuits.

Mixed logic is based on the key observation of DeMorgan's theorem: that logical operations have equivalencies when their inputs and outputs are inverted. DeMorgan's Square shows the equivalencies of the four basic gate types.



AND

A B	
0 0	0
0 1	0
1 0	0
1 1	1

NAND

A B	
0 0	1
0 1	1
1 0	1
1 1	0

NOR

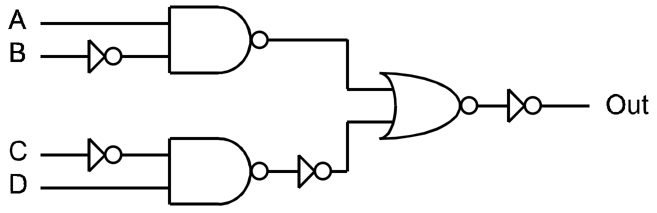
A B	
0 0	1
0 1	0
1 0	0
1 1	0

OR

A B	
0 0	0
0 1	1
1 0	1
1 1	1

Inverting the output of the gate moves horizontally in the square. Moving vertically is accomplished by inverting all gate inputs (turning the truth table upside down). Note that each gate has two DeMorgan equivalent gate symbols. While it may seem counterintuitive to draw an AND gate as an OR body with inverted inputs and an inverted output, this variation makes mixed logic design possible.

Consider the example logic circuit using traditional gate symbols.



This circuit diagram includes both the specification and implementation of the designed logical function. The desired function might be:

$$Out = \overline{A \cdot \overline{B}} + \overline{C} \cdot D$$

Or it might be that the specified NAND and NOR gates satisfied an implementation requirement that is independent of the desired function. If it is necessary to reimplement the function using only NOR gates and inverters, the design process is error prone and often yields confusing and non-optimal implementations.

A mixed logic approach exploits the symbol variation provided by DeMorgan's Theorem to support three design rules:

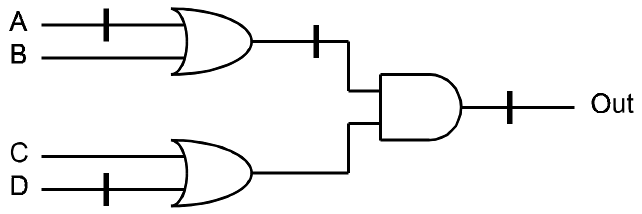
1. All logical operations in the desired logical expression become gate bodies in the circuit.
2. All complements in the desired logical expression become bars in the circuit.
3. All bubbles in the circuit are paired so they *cancel out*. One bubble is attached to each bar.

Using mixed logic design, the desired logical function to be *read* from the circuit regardless of implementation. Moving bubbles around allows an implementation to be changed to meet a specific technology requirement.

To illustrate mixed logic design, consider the following logical function.

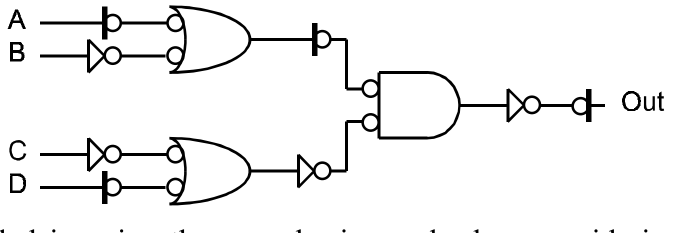
$$Out = \overline{\overline{(\overline{A} + B)} \cdot (C + \overline{D})}$$

The form of the expression is chosen based on the application requirement and is independent of implementation. This expression can be represented graphically.

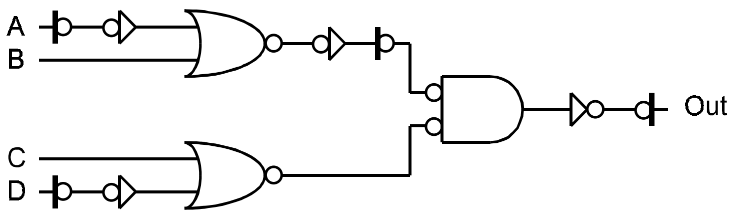


While not an implementation, this circuit shows the logical combination of the inputs. The bars represent where an inversion of the signal is required. This basic structure is unchanged regardless of the gate types used to implement it.

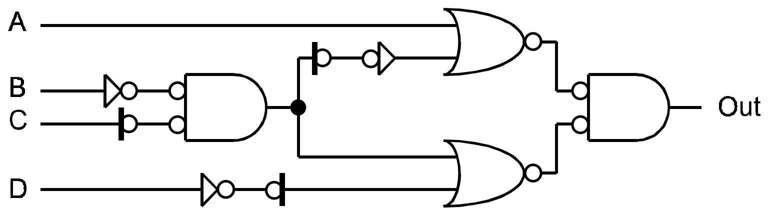
To realize a specific implementation, bubbles are added in pairs along wires until the design gate type is achieved. To produce the required inversion at bars, a single bubble is also left at either the front or back of each bar. When there are no desirable gate bodies to attached a needed bubble, a buffer body is first added and then transformed into an inverter. One implementation of the function employs the gate types used in the first example.



Since bubbles are added in pairs, they can be ignored when considering a circuit's behavior. Bubbled bars, while having no physical manifestation, indicate that the unbalanced bubble it matches creates an actual inversion of the signal. Several buffer bodies are needed to provide attachment points for bubbles while maintaining desired gate types. These buffers become inverters that do have a physical manifestation in the implementation. Note that this design is identical to the first example, but for using DeMorgan equivalent gate symbols. Reimplementing using only NOR gates yields a similar circuit but for the gate type used. Only bubbles and buffers are rearranged.



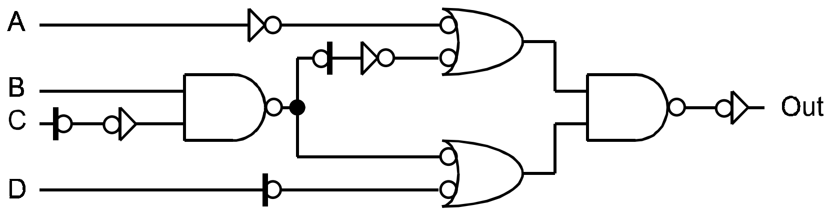
Here's another example.



Given the circuit, the desired logical behavior can be unambiguously extracted.

$$Out = (A + \overline{(B \cdot \overline{C})}) \cdot ((B \cdot \overline{C}) + \overline{D})$$

The fact that this implementation includes only NOR gates neither changes nor distorts the behavior. Note the common subexpression (B or NOT C) used in two places in the circuit. When bubbles are added to a wire that is used multiple places (*fanout* > *1*), multiple bubbles on the consumer end must be added to match the bubble on the producer end. This can be seen when this behavior is reimplemented using only NAND gates.



In summary, mixed logic design:

1. Decouples the behavior of a circuit from its implementation.
2. Supports self documenting circuits.
3. Provides a simple process to reimplement a circuit using different gate types.