Boolean Logic

Using De Morgan's Theorem to Create NAND gate Circuits

As we have seen, we can imitate the behaviour of NOT, AND and OR gates by simply using NAND gates (or alternatively, by simply using NOR gates). This is quite useful because it would actually turn out to be cheaper to build a circuit consisting entirely of one type of gate, thus avoiding component spread – and, if we are designing an integrated circuit, it means that we can just reuse the mask for a NAND gate, rather than having to design different masks for the various types of chip.

So, we need to be able to not only simplify Boolean expressions, so that they use as few gates as possible, but also rearrange them so that they only use NAND gates (or, only use NOR gates).

We do this by applying De Morgan's theorem in a particular way.

De Morgan's Theorem

De Morgan's Theorem states:

/ (A + B) = /A . /B
/ (A . B) = /A + /B

It will actually make things clearer if we use the alternative notation for the NOT operation, which is the horzontal bar, like so:

$$\overline{(A + B)} = \overline{A} . \overline{B}$$

$$\overline{(A . B)} = \overline{A} + \overline{B}$$

A rule of thumb for remembering De Morgan's theorem is:

"Break the bar and change the sign."

Rearranging Boolean Expressions

Let's say that we have the following expression:

$Z = A + B + C$ Bearing in mind that each of the Boolean variables, A, B or C, could itself represent a Boolean sub expression, let's see how we could rearrange using De Morgan's theorem.

If you only have two-input NAND gates available, you would rearrange the expression as follows:

Deal with the first pair of terms first, by putting a double NOT above them.

$$Z = \overline{\overline{(A + B)}} + C$$

Break the bar and change the sign.

$$Z = \overline{(\overline{A} . \overline{B})} + C$$

Now treat the bracket as a single term, and include C by putting a double NOT above them.

$$Z = \overline{(\overline{A} \,.\, \overline{B})} + C$$

Break the bar and change the sign.

$$Z = \overline{(\overline{A} \,.\, \overline{B})}.\overline{C}$$

This gives us a circuit which consists of NOT A & NOT B being fed into a NAND gate. The output from that has a NOT operation applied to it, and is then fed into another NAND gate along with NOT C.

(See Diagram 1)

Resist the temptation to cancel the two NOT operations above the brackets.

So we have gone from using two OR gates, to a circuit which uses 6 NAND gates (Don't forget that the NOT operations can be imitated by using a NAND gate). But, it makes production of the circuit cheaper and easier.

Exercise
Try setting up the circuit for the initial circuit on the simulator, and work out the truth table. Now build the circuit using only NAND gates, and see if the truth tables match.

Let's try a slightly more realistic example:

Let us say that we have taken our sum of products expression from the truth table, simplified it by using a mixture of the algebraic rules and Karnaugh maps, and now we need to rearrange it so that we can build it using only NAND gates.

$$Z = A.\overline{C} + B.C + A.\overline{D}$$

Double NOT above the first pair of terms

$$Z = \overline{\overline{A.\overline{C} + B.C} + A.\overline{D}}$$

Break the bar and change the sign.

$$Z = \overline{(A \,.\, \overline{C})} \;.\; \overline{(B \,.\, C)} \;+\; (A \,.\, \overline{D})$$

Double NOT above the whole expression.

$$Z = \overline{\overline{(A \,.\, \overline{C})} \;.\; \overline{(B \,.\, C)} + (A \,.\, \overline{D})}$$

Break the bar and change the sign

$$Z = \overline{(A \,.\, \overline{C})} \,.\, \overline{(B \,.\, C)} \,.\, \overline{(A \,.\, \overline{D})}$$

(See Diagram 2)

Exercise
Try setting up the circuit for the initial circuit on the simulator, and work out the truth table. Now build the circuit using only NAND gates, and see if the truth tables match.

DIAGRAM 1

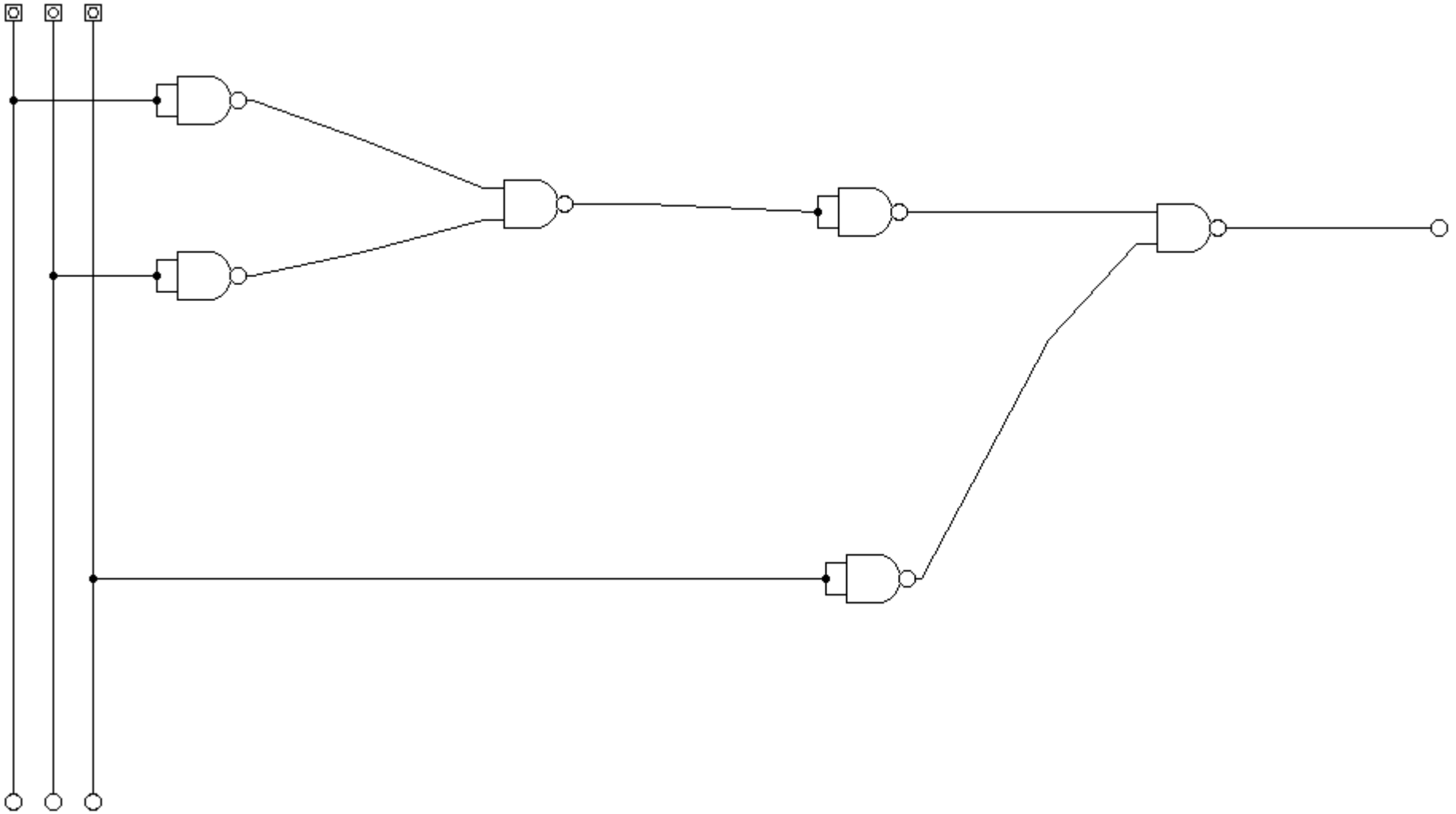

Diagram 2