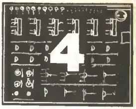
LOGIC TUTOR EXPERIMENTS..



DE MORGAN'S THEOREM

FIRSTLY the answer to last month's question. There are various ways of getting the six input AND but they all use the same principle. There is an Associative rule in Boolean algebra which says that if you have a number of variables coupled by the same logical functions then the variables can be grouped together in sub-groups and combined by their function independently; the independent groups can then be coupled together with like functions to produce the final desired effect.

Thus if we want to AND together inputs A, B, C, D, E and F we can carry out the operation in three stages; firstly we AND A with B and C (as a sub-group) then D with E and F as a separate sub-group. Finally we take the outputs of each of the sub-groups and AND them together in a two input gate to give the total effect.

One form of six input AND gate using NANDS is shown in Fig. 4.1. Notice that it is necessary to use a lot of gates to carry out what is basically a very simple function. It would be much more economical—in space and cost—to use a six input NAND followed by an inverter or alternatively convert a four input NAND into a six input version using an expander before inverting.

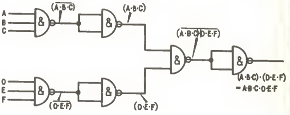


Fig. 4.1. A solution to last month's problem.

DE MORGAN'S THEOREM

Referring to the truth table for the NAND we could say that the output is 1 when A is 0 OR B is 0. Remember we are describing the same function as last month but are using a different point of view. Using the Boolean nomenclature that $\overline{A}$ represents "when A is nought" we can say that the output Q is given by $\overline{A}$ or $\overline{B}$

$$Q = \overline{A} + \overline{B}$$

But from a different view point—last month we saw that

$$Q = \overline{A.B}$$

Therefore by normal algebraic argument we can say that

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

This proves the first of De Morgan's Theorems which—in very simple terms—says that an inverted AND is identical to a sort of inverted OR.

There is a second theorem which is very similar (it is worth you thinking how to argue it out) which says:—

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

Again in simple terms an inverted OR is the same as a sort of inverted AND.

Before moving on, take note of a catch that beginners sometimes fall into. $\overline{A.B}$ is *not* the same as saying $\overline{A}.\overline{B}$. This means that when writing Boolean expressions you have to be careful with the length and breaks of the negate bar over the top of the alphabetical characters. Sometimes brackets are used to make the distinction clear in complex expressions.

OR FROM NAND

De Morgan's Theorem is one of the most used in Boolean algebra because it gives a NAND gate a duality of purpose. Depending on how we want to think we can say that the output is either $\overline{A.B}$ or $\overline{A} + \overline{B}$. We have already utilised the former to give us AND from NAND. Now we can use the latter to give us basic OR.

Fig. 4.2 shows the way of producing a fundamental two input OR function. The inputs A with B become $\overline{A}$ with $\overline{B}$ at the outputs of their respective inverters. The output of the final NAND can be considered to be an OR function coupling the inverted form of its inputs. Therefore the output in this case is $\overline{\overline{A}} + \overline{\overline{B}}$. The double negates over each variable cancel and we are left with A + B.

Use the toggle switches on the Logic Tutor to provide inputs to this circuit and monitor the logic levels at the various nodes on the lamps and check these against the truth table for the circuit shown in Fig. 4.2.

As an exercise try and use the knowledge you now have of the Associative rule and the gates available on Logic Tutor to produce a six input OR. (One answer to be given next month).

by M. Hughes

Next month we shall deal with the EXCLUSIVE OR function.

This column also describes $\overline{A} + \overline{B}$

A	B	$\overline{A.B}$
0	0	1
1	0	1
0	1	1
1	1	0

A	B	$\overline{A}$	$\overline{B}$	A+B
0	0	1	1	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	1

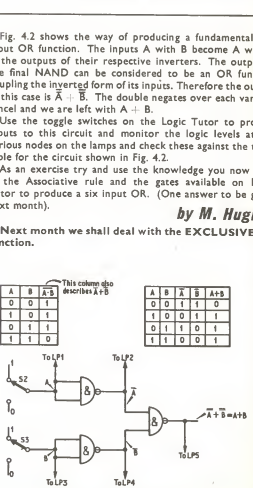


Fig. 4.2. OR from NAND logic.