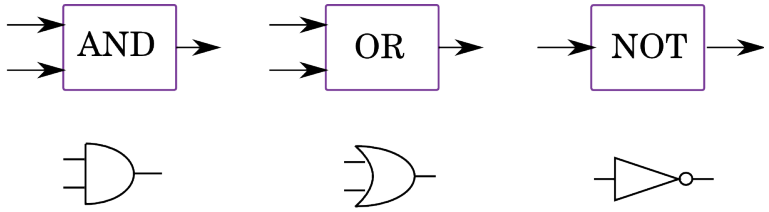
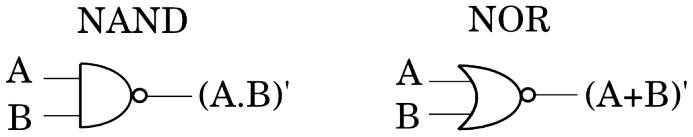
Lecture 2: Boolean Functions, Gates and Circuits

While Boolean algebra is the fundamental formal system for digital circuit designers, digital circuits are their final product. Digital circuits are similar to Boolean block diagrams but each block is replaced by an easily recognizable graphical symbol called a gate. A circuit is designed by connecting gates together. Since the gate symbols are clearly recognisable there is no need to label them AND, OR etc., and they can be composed into more complex building blocks which in turn are given their own symbols. The basic gate symbols are:

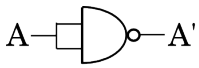


The NOT gate is often referred to as an inverter gate. Actually, the inversion is indicated by the little circle on the right hand end. The triangle on the left hand side indicates which is the input and which is the output. The circle can also be attached to the AND and OR gates to make up two new gates which are called NAND (not and) and NOR (not or).

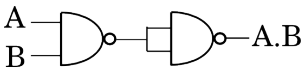


These gates are the fundamental building blocks of digital circuits. In fact it would be possible to build any digital circuit using just NAND gates or just NOR gates. To demonstrate that this is so we will use the NAND gate to build implementations of all the other gates that we have seen so far.

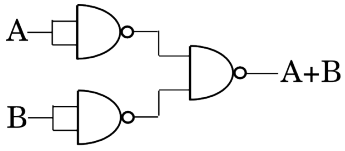
Firstly, the inverter is easily implemented by connecting the two inputs of the NAND gate together. In terms of Boolean equations we can write $(A \cdot A)' = A'$. The circuit is:



Having created an inverter we can now implement the AND gate simply by inverting the output from a NAND gate. In terms of Boolean equations we can write $((A \cdot B)')' = A \cdot B$. The circuit is:

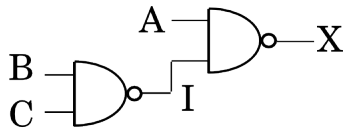


In order to create the OR gate we need use de Morgan's theorem: $A + B = (A' \cdot B')'$. The circuit is created by using two inverters to form A' and B' and then one more NAND gate.



Finally we can create the NOR gate simply by adding another inverter stage to the OR gate implemented above. As an exercise you can try to design all the other gates using just NOR gates.

We can build circuits of any complexity out of these simple building blocks. We can look at their function in two different ways. Firstly we can write down a boolean equation by tracing through the circuit, and then try to simplify it. Secondly we can calculate a truth table which completely specifies its behaviour. This is most easily done by calculating the intermediate nodes. Consider a circuit made up by cascading two NAND gates.



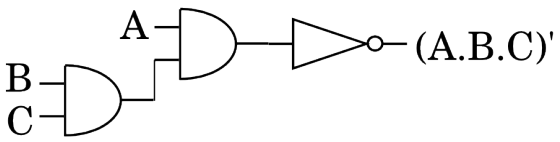
We mark the output as X and the intermediate wire as I. We can use Boolean algebra to write:

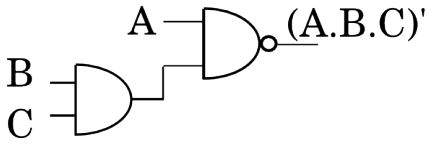
$$I = (B \cdot C)'$$
$$X = (A \cdot I)' = (A \cdot (B \cdot C)')'$$
$$X = A' + B \cdot C \text{ (by de Morgan's theorem)}$$

We can build a truth table by systematically working through the circuit. We calculate I first, and then use it to find X.

A	B	C	I = $(B \cdot C)'$	X = $(A \cdot I)'$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

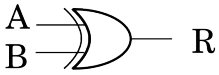
As another exercise we can consider how to build a three input NAND gate. In Boolean algebra we write this as $X = (A \cdot B \cdot C)' = (A \cdot (B \cdot C))')$. Notice that we are using the associative rule of Boolean algebra to build a three input AND gate from two two input AND gates. The final circuit can be simplified by using a NAND gate to replace the AND and inverter combination.





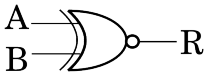
We introduce two very commonly used gates which have their own individual symbols in Boolean circuit design.

Exclusive Or (XOR)



R = A.B' + A'.B

Exclusive Nor (XNOR)



R = A'.B' + A.B

From the boolean equations we see that it is possible to build these out of other gates, but in practice they are so useful that they deserve to be considered Boolean functions in their own right. The truth tables are as follows:

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

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

We have now seen one one input gate and six two-input gates, but are these the only useful ones? One may ask the question "how many different gates one can build?" and "what do they all do?" We can answer the first question by noting that with one input, there are two possible outputs, and therefore $2^2 = 4$ different gates. With two inputs, there are four possible outputs, and therefore $2^4 = 16$ different gates. If we consider the case of single input circuits, we can enumerate the four possibilities:

A	G0	G1	G2	G3
0	0	0	1	1
1	0	1	0	1

We see that G2 is the inverter gate, and is the only useful one of the four. G0 always produces the output 0, G1 simply returns the input value of A, and G3 always produces the output 1. The function of each gate can be summarised by the following table:

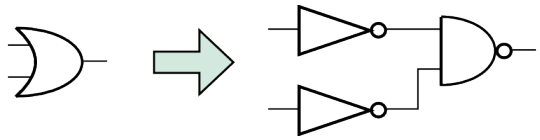
	G0	G1	G2	G3
R =	0	A	A’	1

By a similar method we can enumerate all possible two input gates and their Boolean functions:

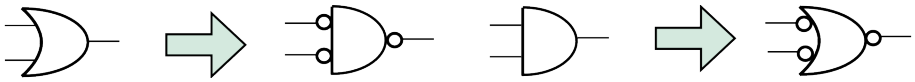
AB	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
R =	0	AND		A		B	XOR	OR	NOR	XNOR	B’		A’		NAND	1

Again we see that the six gates we have already described are the ones that provide useful functions of the two inputs. We may find a use for the others in specific design problems, but they are not sufficiently useful to merit a special symbol or function name.

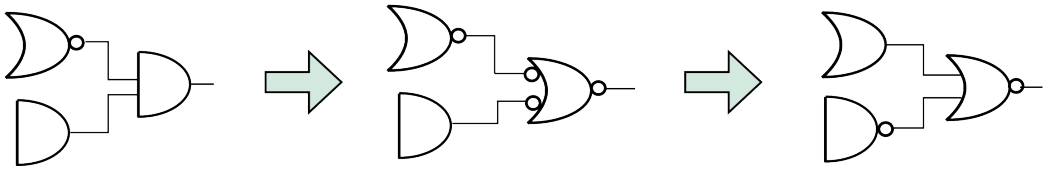
Graphical manipulation of circuit diagrams can often be done to simplify a design. Consider de Morgan’s theorem which we can use to write the equation: $A + B = (A' \cdot B')'$. This could be the basis of a circuit manipulation that we can make simply by looking at the circuit diagram:



Often it is convenient to use just the circle for inversion rather than the whole inverter gate symbol. Doing this we represent de Morgan’s theorem graphically as follows:



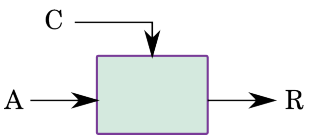
And we can now perform the same manipulations that we did using Boolean algebra directly to circuits as we design them:



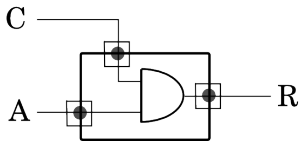
Apply de Morgan's theorem

Re-group and cancel the inverters

To finish up this lecture we introduce a new way of looking at Boolean circuits. So far we have treated gates as implementing Boolean functions of their input variables, and have not considered the possibility that the inputs may be of different types. We now introduce the idea that Boolean variables in circuits may belong to one of two types: data and control. Control variables are used to determine the *flow* of data. Here is a simple example. The block diagram shows a circuit with one control variable, one data variable and one output.



We now specify the action of the circuit informally as: If the control variable is 1 then the output is the same as A. If it is zero the output is zero. In other words the circuit either allows the data to pass from input to output or blocks it, depending on the control variable. The implementation of this circuit is trivial: it is just the AND gate, We now see why the name *gate* is used.

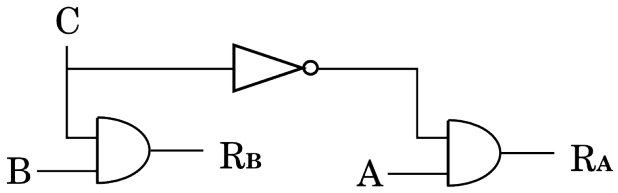


Thinking about a circuit operation in this way will prove invaluable as we begin to design more complex digital circuits.

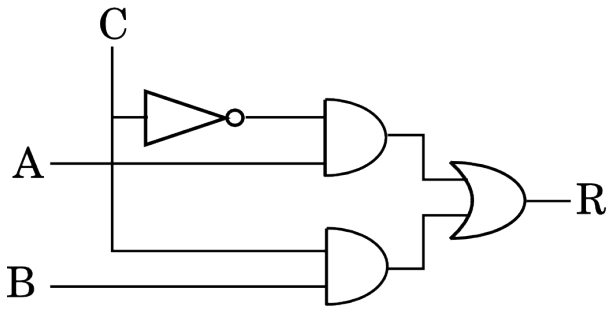
We can extend this concept to design a very useful circuit. It has two data inputs, A and B, one control input and one output. The specification is: "If the control input is zero then the output is the same as A. If it is 1, then the output is the same as B. In other words it is a switch that choses between data line A and data line B. Starting with this specification we can express the circuit's function as a truth table:

C	A	B	R
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

We will introduce a systematic method for designing circuits from truth tables next lecture, but for the moment we will do the design by reasoning. We have seen that the AND gate can be used to block a data path, so we can apply this idea to the data inputs A and B, but invert the control input so that when C is 0 A is unblocked but B is blocked and vice versa. We can do this with just two AND gates and one inverter.



All we need to do is to combine the two outputs into one. We note that in Boolean algebra $A + 0 = A$, and so all we need to do is to combine R_A and R_B with an OR gate.



The final circuit is called a *multiplexer*, and as we shall see is one of the most useful building blocks in digital computer design.