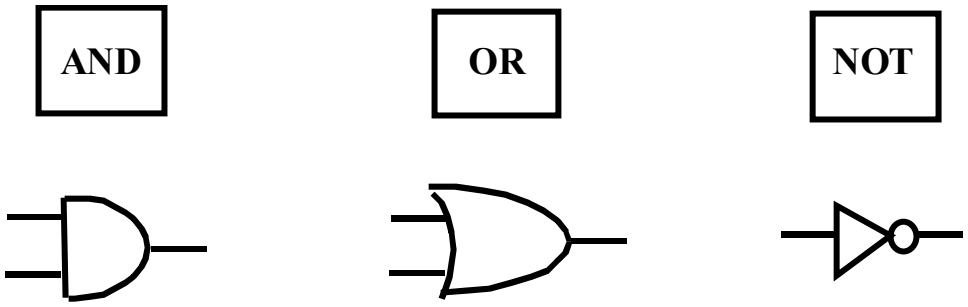
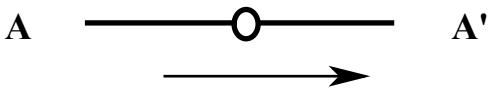
Department of Computing
DOC 112 - Computer Hardware

Lecture 2: Gates, Integrated Circuits and Boolean Functions

While Boolean algebra is the fundamental formal system for digital circuit designers, digital (i.e. binary) circuits are their main tools. Digital circuits are similar to Boolean block diagrams but each block is replaced by an easily recognizable graphical symbol or **Gate**. Not only this makes the operator words **AND, OR** and **NOT** redundant (and therefore unnecessary), but also this allows the expansion of the "operators" into a larger class of gates. The basic fundamental operations and equivalent gate symbols are:



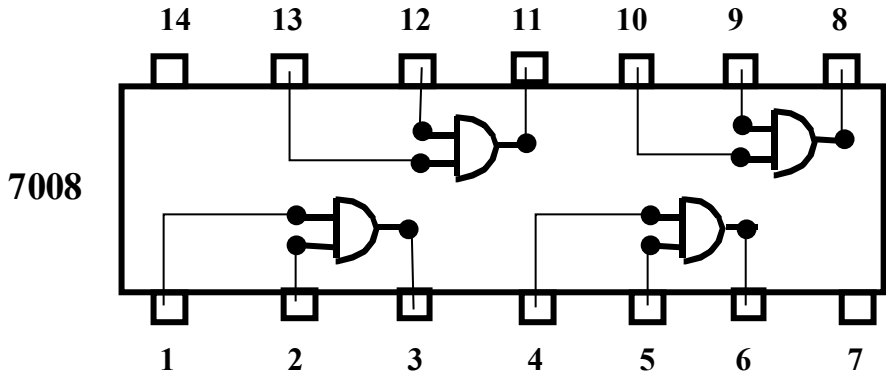
The **NOT** gate is called an **Inverter** gate in digital gate terminology. Actually, the little circle symbol is all that needed for "inversion" of a Boolen value but it is unusual that the circle is used by itself since then it is impossible to tell which is the input and which is the output.



The circle symbol on the other hand, can be attached to any gate symbol whose output is then inverted. In this manner we can define two new gates: **NAND** and **NOR**.

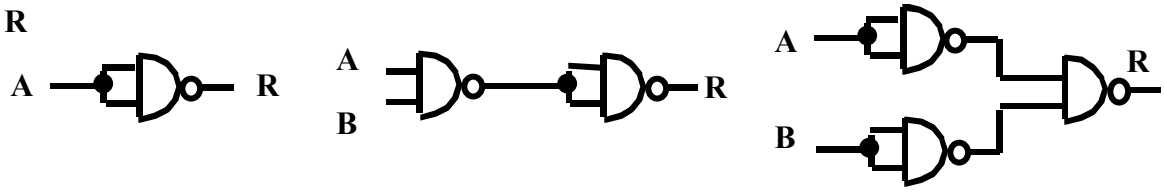
NAND = (A•B)' **NOR = (A+B)'**

The significance of these gates is that they are the fundamental building blocks of most practical digital circuits. If one wants to build a relatively small digital circuit, one is most likely to use a digital circuit board with sockets into which ready-made **Digital Integrated Circuit** or simply **IC** devices can be inserted. Today most simple ICs are already standardised and the most common ones belong to the so-called **7400** series. The simple ones are made by **Small Scale Integration** technology (or **SSI**) and as an example, the schematic diagram of the **7408** device is shown on the next page.



As shown, this IC has fourteen pins out of which twelve are used for four two-input AND gates and two are used for supplying power to the device (V_{supply} = 5 volts, and Ground = 0 volts). Most simple SSI IC devices have either 14 or 16 pins. We will work with these in the tutorials and they will be also used (on paper) in larger design exercises.

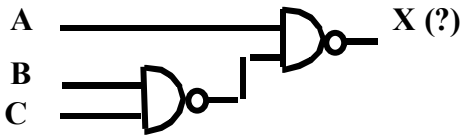
Since ICs come with given number and type of gates it is not immediately obvious how one could build an arbitrary digital circuit. However, we will show now that by using inverters and DeMorgan's theorem one can transform AND gates to OR gates and vice versa. In fact, a two-input NAND gate is all that is needed to build any digital circuit. We can show this by building an Inverter, an AND and an OR gate purely from NAND gates.



R = (A•A)' = A' **R = ((A•B)')' = A•B** **R = (A'•B')' = A + B**

Finally, often we need more than two inputs for realising terms like (A+B+C+D). Cascading gates can provide multiple inputs. Lets see what happens if we cascade two two-input NAND gates, what kind of three-input device do we get?

X = (A•(B•C)')' = A' + B•C



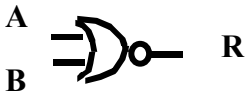
A B C	(B•C)'	(A•(B•C)')'
0 0 0	1	1
0 0 1	1	1
0 1 0	1	1
0 1 1	0	1
1 0 0	1	0
1 0 1	1	0
1 1 0	1	0
1 1 1	0	1

By Boolean algebra **X = ((A) • (B•C)')'** and applying DeMorgan we **do not** get a three-input **NAND** gate.

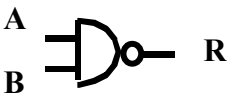
So, how can we create a three-input **NAND** gate?

$$R = (A \bullet (B \bullet C))' = (A \bullet B \bullet C)'$$

DeMorgan's law can also be used to show that a NAND gate with inverted inputs is equivalent to an AND gate and the same gate transformation works for the NOR gate (remember duality?).

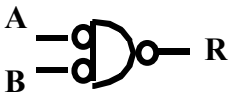


$$R = (A+B)' = A' \bullet B'$$



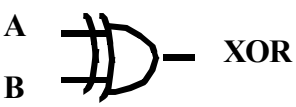
$$R = (A \bullet B)' = A' + B'$$

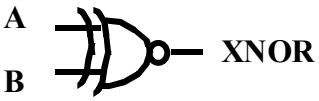
$$R = (A'+B')' = A \bullet B$$



$$R = (A' \bullet B')' = A + B$$

There are two more useful two-input gates called the **Exclusive OR** or **XOR** and the same with inverted output; i.e. the **Exclusive NOR**, or **XNOR** gates. We show the symbols and truth tables for these gates below. We will get familiar with these gates later on in the course.

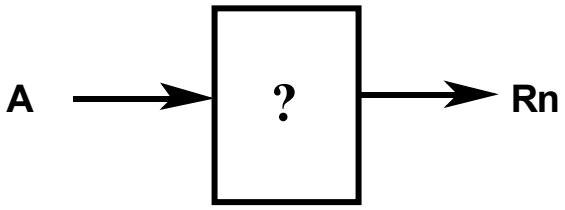




A B	XOR
0 0	0
0 1	1
1 0	1
1 1	0

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

We have now looked at one one-input and six two-input gates. One may ask the question: How many different gates one can build? With other words, how many different truth tables can we produce for an arbitrary gate with a given number of inputs? We show below that for a one input gate there could be four ($(2*1)^2 = 4$) possible truth tables:



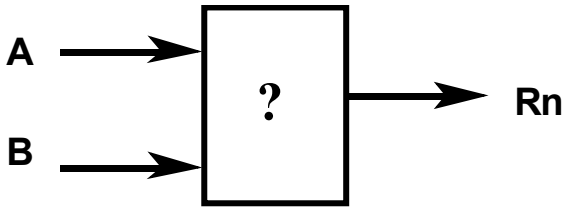
A	R0	R1	R2	R3
0	0	0	1	1
1	0	1	0	1

The fourth column, or the truth table values indicated by **R2** agree with the inverter or **NOT** function. Instead of considering these truth tables only as a list of ones and zeros, we could look at the **functional** relationship between the input variable **A** and the output **Rn** (the truth table values). We already know that for output **R2** we have the functional relationship **R = A'**. How about the others?

The first output column, or column **R0** does not change so it is a **constant**. In fact, it is the constant **0** (**R = 0**). Column **R1** is simply **R = A**. And finally, column **R3** is the constant **1**. We can provide now the same information in functional form:

	R0	R1	R2	R3
A	0	A	A'	1

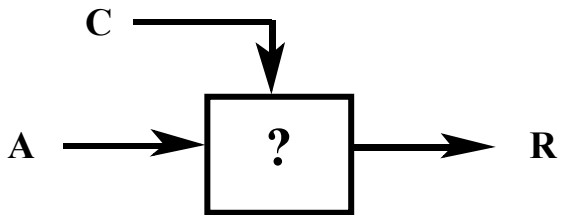
The functional description is an extremely important way of looking at digital systems. Now we may look at the possible truth tables for a general gate with two inputs. There are $(2*2)^2 = 16$ different possible truth tables:



AB	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	
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
	0	AND	A	B	XOR	OR	NOR	XNOR		B'		A'		NAND		1

We can already recognise many of the output combinations as known digital functions:

Now we examine a new idea: **Functional Description with Control Variables** or the idea of a variable function generator block. Let us modify the two-input gate so it looks like this:



Only the interpretation (and symbols) of the input variables has changed, this is still a two-input one output digital circuit. However, functionally, this circuit has one input variable, one output variable, and one control variable. It is assumed that the control variable is set either to **0** or **1** and then the circuit becomes a one-input, one-output device, providing the function **R = f(A)**.

The control variable now can choose between two possible functions. Thus we can write:

$$R = f_{(C)}(A)$$

For example, we could chose between the unaltered input variable **A** or its complement **A'**; thus we have:

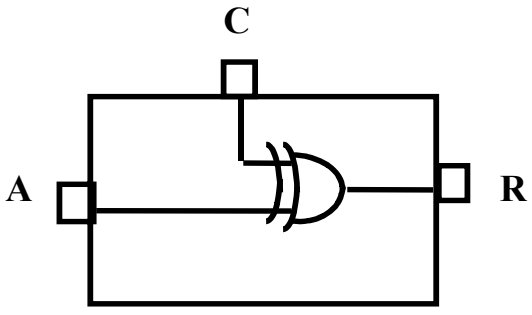
$$f_{(0)}(A) = A \qquad\qquad f_{(1)}(A) = A'$$

And the truth tables are:

A	R = f(0)(A) = A	R = f(1)(A) = A'
0	0	1
1	1	0

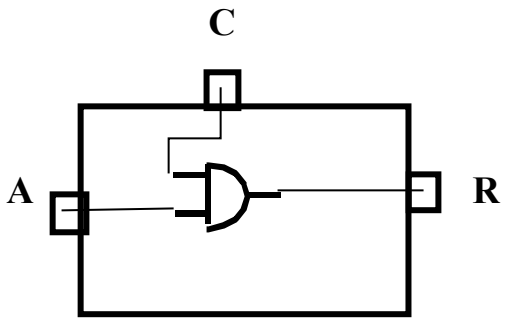
The truth table can be rearranged to indicate a general two-input one-output device:

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



A simple **AND** gate can also be used as a controlled Boolean function generator:

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

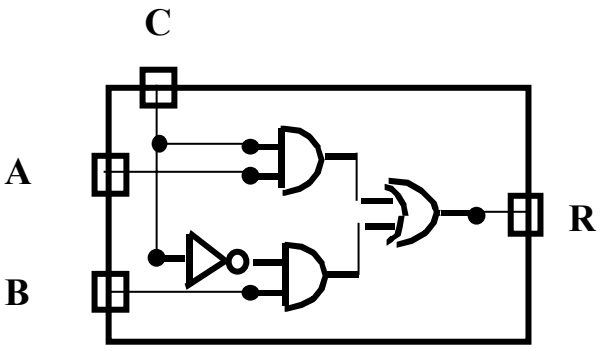


Therefore:

$f_{(0)} = 0$ **(nothing)** $f_{(1)} = A$ **(the unchanged input)**

Thinking in terms of functionality, we can see that an **AND** gate used as a controlled gate stops the signal to go through when the control signal value is **0** and it lets it go through unaltered when the control signal value is **1**.

From the **AND** gate functions we can build a **multiplexer** which will become extremely useful later in the course.



C AB	R	
0 00	0	
0 01	1	B
0 10	0	
0 11	1	
1 00	0	
1 01	0	A
1 10	1	
1 11	1	

The multiplexer has been designed on functional lines. The outputs of two controlled **AND** gates are connected to an **OR** gate. When the control input's value is **0,** it disables the output of the upper **AND** gate (input **A**) and allows input **B** to go through. When it's output is **1** then it is the reverse and input **A** goes through. It behaves as a **switch**.