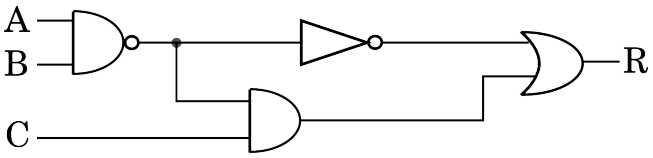
Lecture 4: From Problem Description to Circuit

In the last lecture we saw how combinational digital circuits can be built once their input/output function had been fully defined, either by a truth table or by a canonical Boolean equation. The simplest method is to use an AND gate for each *one* in the truth table or minterm in the canonical equation. The outputs of the AND gates are then connected to a many-input OR gate. Thus, we could build a general device of any number of inputs (within reason) which could realise an arbitrary number of minterms.

Let us look at the following circuit and its truth table:

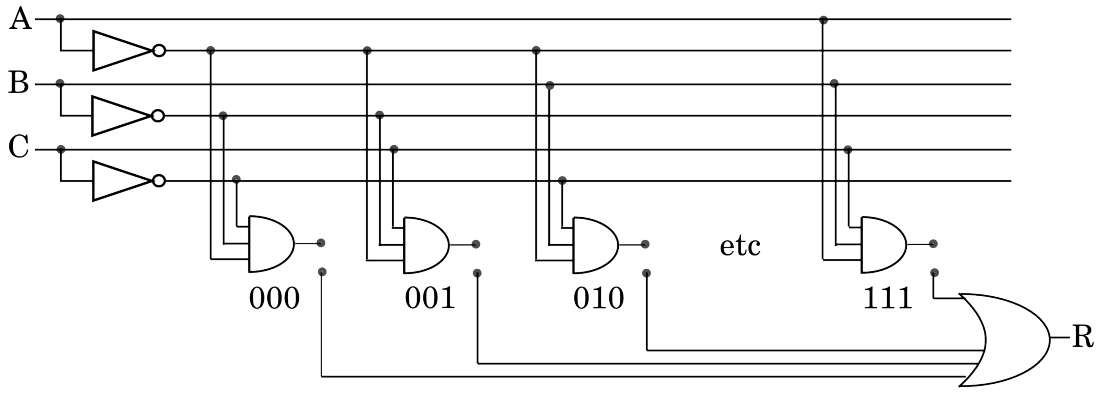


A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

By looking at the *ones* in the truth table, we can write down the canonical Boolean equation by inspection:

$$R = A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C$$

Thus this circuit can be built from five three-input AND gates and a five-input OR gate. However, a more general circuit shown below can generate all possible three-input digital circuits.



This is called a Programmable Array Logic (PAL) device and for the three-input circuit has eight three-input AND gates (the number of possible functions) and one eight-input OR gate.

The PAL device comes with so-called "fusible links". Gaps are shown at the output of the three-input AND gates. As long as they are left alone, they provide no signal to the OR gate and the "unprogrammed" output of this device is always equals to Logic-0. The device may be "programmed" by special equipment which sends a large current through selected links and "fuses" them so that they will conduct and thus any selected AND gate (corresponding to the minterm of the truth table) could provide a 1 to the OR gate and make the output of the OR gate equal to 1. Notice that it makes no difference what the input values are at any one time, only one AND gate's output will be a 1. (Again, this is how minterms are defined!).

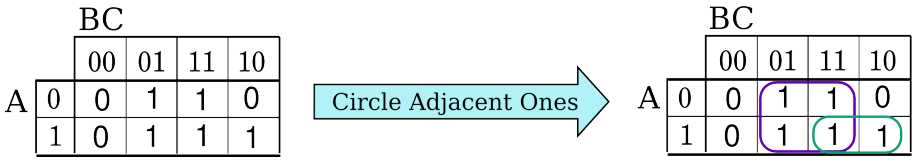
Let us go back to our original circuit, which had only four gates of various kinds. We started with a relatively simple circuit and by using our general method ended up with a much more complicated one. If we use a PAL device then we always end up using the eight three-input AND gates and one eight-input OR gate built into the PAL. This is clearly not a minimal solution, but may be of use when we wish to make fast prototypes of hardware designs, or even to create a design automatically by computer.

Another observation may be made about the truth table. There are five *ones* but only three *zeros*. Can we use the *zeros* and have a simpler circuit? Of course we can! There is duality and we should never forget it. Using duality we can write down (by inspection) the alternative canonical Boolean equation which is a product of maxterms:

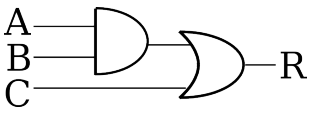
$$R = (A + B + C) \cdot (A + B' + C) \cdot (A' + B + C)$$

Using this form we want to ensure that for all inputs which provide 0 output, there is at least one of the bracketed terms which is equal to 0. The first such input is equal to 000, and for that input combination only, (A+B+C) will will be zero. This circuit will require a three-input AND gate and three three-input OR gates. However, if we use a general purpose programmable logic array for implementation we don't make any saving in gates, since it will provide a three input OR gate for each of the eight possible maxterms.

However, as we have seen from the last lecture, if we are doing a custom design we can make the circuit much smaller, and this will save space on our silicon chip and allow us to place more functionality on it. For small problems we can using a Karnaugh-map (or K-map) to spot factorisations in our Boolean equations. For larger problems it is possible to do this algorithmically, but we will not cover that topic in this course. In this case the Karnaugh map is:



And this produces the following minimal equation and circuit:



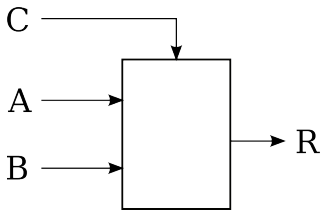
$$R = C + A \cdot B$$

Could you have guessed that we would have such a simple implementaqtion by looking at the original circuit? Possible, but in general it may not be so easy.

Now we will look at a practical problem from the point of view of building a cost-effective digital circuit. We will be particularly interested in finding a reasonably inexpensive solution to a specific digital problem. If we are going to manufacture our circuit we will need to consider minimising the size of the circuit on a silicon chip, and as we shall see this is not simply a matter of finding the circuit with the minimum number of gates. The design exercise is very similar to the one that you will be given in the coursework.

Design Exercise 1

The circuit is to have three inputs and one output. One of the inputs is specified as a control input (we shall designate it C). When this control input is at the logical 0 value, the output is the logical (or Boolean) AND function of the other two inputs. When C is at the logical 1 level, the output is equal to the Boolean OR function of the other two inputs.



Control	R
0	A.B
1	A+B

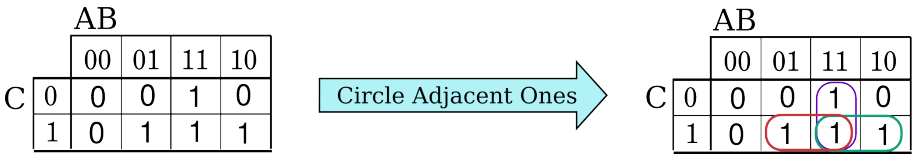
Step 1 Generate the Truth Table

The first step of the design is to translate the verbal description of the problem into a truth table. In this case, this can be done by inspection:

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

Step 2 Generate the Karnaugh Map

The next step is to minimise the four-term canonical minterm expression, either by Boolean algebra manipulation or by using Karnaugh maps. It is a small problem, so we can use the Karnaugh map. Be careful whe filling up the map! The order of the entries in the map follow $00 \rightarrow 01 \rightarrow 11 \rightarrow 10$ which is not the order in the truth table.



Step 3 Find the minimal Boolean Expression

The minimum number of circles we need to cover all the *ones* is three. Remember that we cannot group three adjacent *ones* on the Karnaugh map as it won't represent a factorisation. Notice also that it doesn't matter if we cover a square more than once. The circles we have drawn correspond to the terms in the equation:

$$R = B \cdot C + A \cdot B + A \cdot C$$

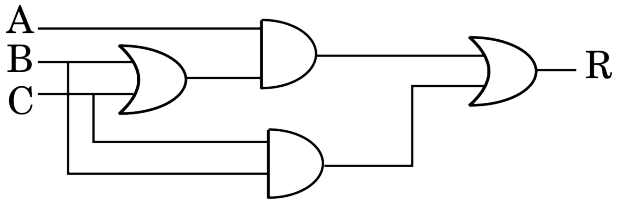
However, this is still not the minimal form since we have some common factors. Unlike the factorisations found by the Karnaugh map these do not eliminate variables from terms, however they can still reduce the number of gates we need for the implementation. Factoring A out of the last two terms gives us:

$$R = A \cdot (B + C) + B \cdot C$$

This is as small an expression as we can get.

Step 4 Draw the Circuit

The circuit that corresponds to our minimised equation is:



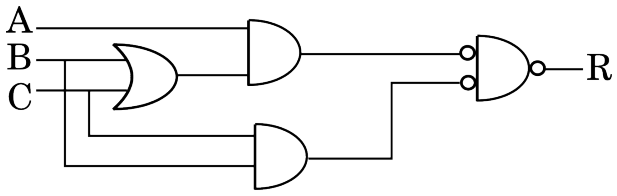
Step 5 Minimise to suit the production method

We have now gone as far as we can go on a theoretical design basis. However, we can make further minimisations that will lead to lower cost, depending on the way we intend to construct the circuit. One possibility would be to go to Maplin electronics and buy some digital integrated circuits. A typical one of these has four two input NAND gates in a black plastic casing with fourteen connectiion pins (one pin for the supply voltage and one for the ground). If we were using this method the design would not necessarily be ideal since we would need an integrated circuit containing AND gates and one containing OR gates, and we would not be able to

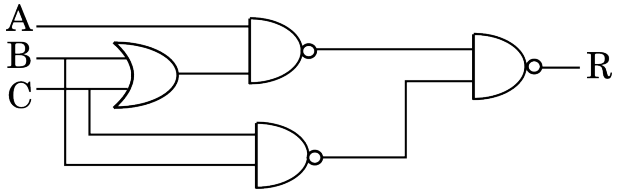
use all the gates in each package. On the other hand, if we were to send our design to a silicon foundary, and manufacture our own integrated circuit, this would not be a limitation since we can connect up whatever gates we need. However, we would have a different criterion to minimise, and that is the area occupied by our circuit on a piece of silicon. The smaller the area the cheaper our circuit will be. Unfortunately, not all gates occupy the same area of silicon. Here is a table of their typical relative areas.

Gate	Nominal Size
INVERTER	3
NAND, NOR	4
AND, OR	6
XOR, XNOR	8

NAND and NOR gates are quite a bit smaller than AND and OR gates, so let's see if we can transform the circuit to make better use of them. Recall that we had a graphical technique for applying de Morgan's theorem to individual gates. The first idea we can try is to apply that to the final OR in the circuit.



We adjust the positions of the inverters and we have converted three of our four gates to NANDs - a considerable saving in space.



We could also use the same technique on the remaining OR gate and convert it to a NAND. However this would mean that we would need another two inverter gates at it's input, and so the are of silicon used would increase. However, if we were planning to build our design from digital integrated circuits we might choose to do this to achieve the best utilisation of the packages.

Step 6 Testing the circuit

Testing is a very important part of digital design. Although it is possible to build prototype circuits, or use programmable gate arrays to test prototype designs, the usual process is to simulate the circuit. As an example of the complexities of modern ICs, the new Intel Core i7 hosts approx 700×10^6 gates! With dedicated circuits performing ever more complex functionalities (eg modern CPUs) it becomes of paramount importance to validate the design against given specifications. This is typically done by systematic testing and formal verification. In testing we essentially aim to check that the output of the circuit as implemented is correct against its intended specification. In formal verification behaviours of parts of the circuits are abstracted against intended inputs and verified by means of highly-sophisticated logic-based techniques. Finding bugs before release is of fundamental importance as addressing the problem after the circuit has entered production can be very expensive (witness the Floating Point Division Bug in the Intel Pentium P5 in 1994). The Intel Core i7 is the first CPU where the execution module has been validated only by formal methods (as opposed to testing).