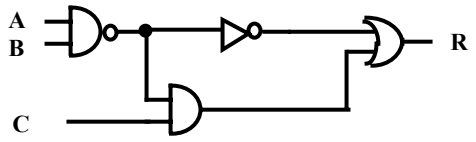
Department of Computing Course DoC 112 / Hardware
Lecture 4

In the last lecture we have seen how combinational digital circuits can be built from their input/output function, i.e. from their truth table. The simplest method is to substitute an **AND** gate for each **1** in the truth table (minterm) with the inputs coming either directly from the circuit input or their inverted value. The outputs of the **AND** gates are 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:

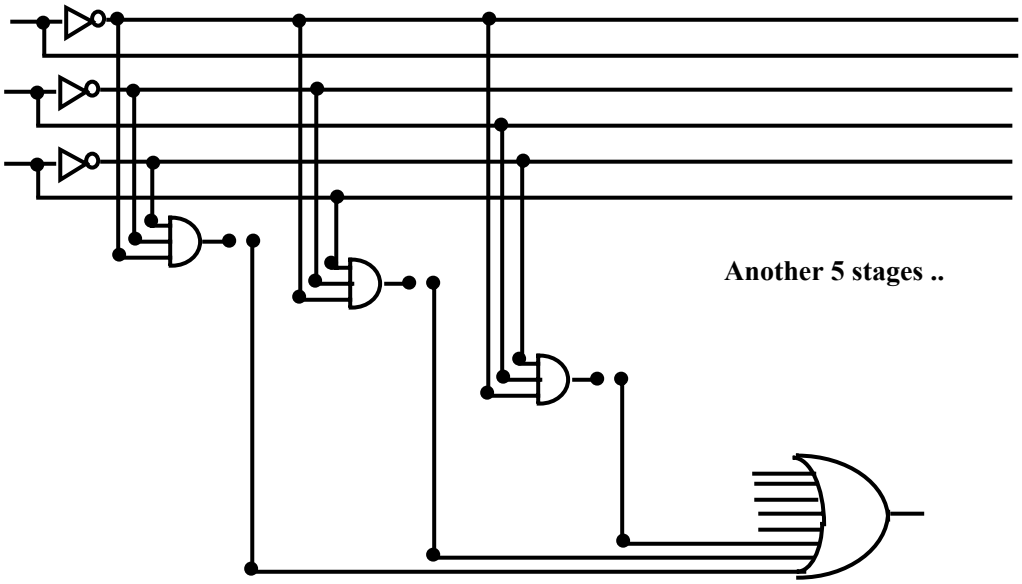


ABC	R
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

The truth table has five **1**s therefore we can write the Boolean expression by inspection to be:

R = A'•B'•C + A'•B•C + A•B'•C + A•B•C' + A•B•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 end up using the eight three-input **AND** gates and one eight-input **OR** gates built into the **PAL**. However, if we use separate **IC**s, a solution with lots of gates is expensive, and we should try to simplify it.

Another observation from the truth table may be made. There are five **1**s (five **AND** gates) but only three **0**s. Can we use the **0**s 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 following Boolean product of **maxterms**:

R = (A+B+C) • (A+B'+C) • (A'+B+C)

It is a bit tricky because we want to make certain 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**; therefore, **(A+B+C)** will is zero, and so on. This circuit will require a three-input **AND** gate and three three-input **OR** gates. Now, this is a bit better!

However, we have seen from the last lecture that we can do even better. We can use a **Karnaugh-map** or **K-map** to do our minimization (at least up to five variables). In this case we have:

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	1

Indicating convenient groupings, we get:

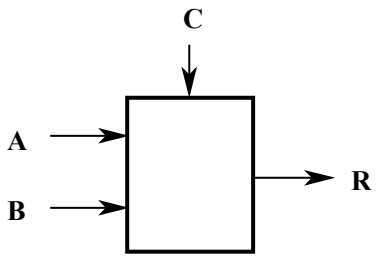
A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	1

or: **R = C + A•B** and end up with two gates, a two-input **OR** and a two-input **AND** gate. Hurray! Could you have guessed this by looking at the circuit?? Maybe -- but it is not 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 with the use of real **IC**s. We are restricted to use only three types of **IC**s: the **7400** (four two-input **NAND** gates), the **7406** (six inverters) and the **7408** (four two-input **AND** gates). Thus, we have restrictions as to what gate types to use and in addition to minimising for the number of gates we use, we must be careful to ensure that we use the smallest number of **IC**s for the circuit. This will require some gate transformations which will be demonstrated below.

Design Exercise 1

Build a three-input, one-output digital circuit. One of the inputs is specified as a control input (we shall designate the control input as **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. You may use only two-input **AND**, **NAND** and **Inverter** gates.



Control

C	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. I prefer the map, so here we go:

(Be careful! The order of the entries in the map follow **00->01->11->10** not the order in the truth table!).

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Step 3 Minimize the Boolean Expression

The next step is to group together neighbouring ones (or zeros) to produce a simpler expression. In this case, there are the same number of ones and zeros, no "don't care" terms and it is most likely that minimising for ones or zeros will produce similar results; therefore, we will do it only for ones.

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

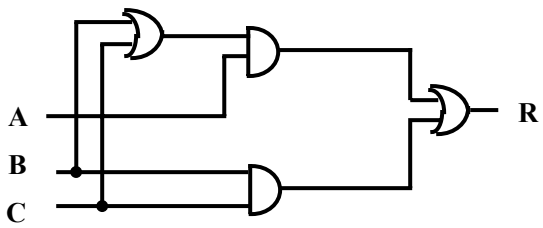
We get: **R = B•C + A•B + A•C**

At this point we examine the minimised Boolean equation whether common factors could be collected and brackets put back into the equation since this will reduce the number of required gates. Here we can write:

R = A•(B+C) + B•C

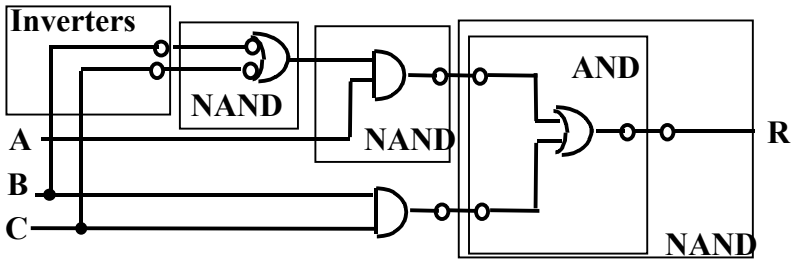
Step 4 Draw the Circuit

We draw the circuit in its minimised form.



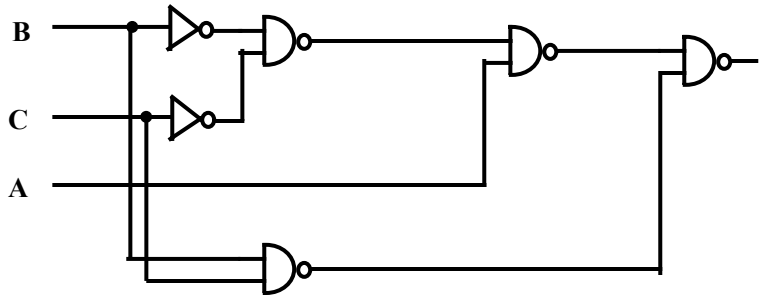
Step 5 Transform Gates for The Minimum Number of ICs

We assign **IC**s, build and then test the circuit. However, the circuit is in a mixed form; i.e. it has both **AND** and **OR** gates but we do not have **OR** gates in our IC repertoire. Therefore, the **OR** gates have to be changed to **AND** (or **NAND**) gates before we could build the circuit.



We can do this by applying DeMorgan's formula; i.e. **(A'+B')' = AB**. However, we do not have to go back to the Boolean equation, we can do this gate transformation operation on the circuit diagram, as shown above. We introduce two inverters (which cancel each other) at the appropriate places in the circuit and transform the gates accordingly.

Moving the remaining circles to the outputs of the **AND** gates turn them into **NAND** gates so we end up with four **NAND** gates and two inverters. And the final circuit is:



<u>Step 6</u> Build the Circuit, Test it and Calculate its Cost

This circuit can be built from two ICs, a type **7400** (four two-input **NAND** gates) and a type **7406** (six inverters). The total cost of the circuit is calculated by a formula that takes into account "**socket costs**" and "**gate costs**". Whenever an **IC** device is used on a circuit board, a space must be reserved for it and possibly a socket must be provided. All these add cost to the final completed circuit board. Even if only one inverter gate is used in a type **SN7408** package of six inverters, the cost associated with the IC package must be included in the total cost.

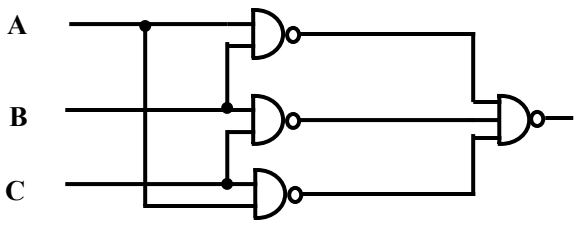
When an **IC** package is not completely utilised (i.e., not all the gates in it are used), the "gates costs" should reflect this fact. If only one gate out of four is used, this should result in a cheaper circuit than when all gates are used. The justification for this cost decrease is that another, unrelated circuit also placed on the same circuit board may need an extra gate, which then could be supplied. This is not an exact calculation but when most gates of all **IC**s are utilised, it does give a good approximate answer.
Thus, the formula for cost calculation is:

Each IC used = 0.5 + U*0.48 (£s) U = utilisation factor

Where 0.5 is the "socket costs", 0.48 is the "gate costs" and **U** is the utilisation factor of the IC and is equal to the number of gates used in the IC divided by the total number of gates in the IC. In our example, the two **IC**s give:

Cost = [0.5 + (4/4)*0.48] + [0.5 + (2/6)*0.48] = £ 1.64

This seems to be a reasonable price and a simple circuit. However, it is very difficult to prove that this is the least expensive circuit. For example, we find out that a different type of IC, type **SN7411** has three three-input NAND gates. Therefore, we can realize the Boolean equation R=AB+BC+AC which we had before, by collecting terms. Using the **SN7411** we produce the circuit shown below which uses three 2-inputs NANDs and one 3-input NAND:



And the total cost is [0.5 + (3/4)*0.48] + [0.5 + (1/3)*0.48] = £ 1.52

The three circuit diagrams of the three **SN7400** type integrated circuits, which are used in this lecture and also in the coming assessed course work assignment, are shown below: